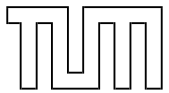
Fortgeschrittene Netzwerk- und Graph-Algorithmen

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Wintersemester 2007/08



Übersicht

Vorlesungsdaten

- Modul: IN2158
- Bereich:
  Informatik III (Theoretische Informatik)
- Semesterwochenstunden:
  4 SWS Vorlesung + 2 SWS Übung
- ECTS: 8 Punkte
- Vorlesungszeiten:
  Montag 12:30 - 14:00 Uhr (MI Hörsaal 2)
  Donnerstag 15:30 - 17:00 Uhr (MI Hörsaal 2)
- Übung:
  Freitag 14:00 - 15:30 Uhr (MI Praktikumsraum 03.09.034)

Dozent

- Dr. Hanjo Täubig
  (Lehrstuhl für Effiziente Algorithmen / Prof. Mayr)
- eMail:
  taeubig@in.tum.de
- Web:
  http://www14.in.tum.de/personen/taeubig/
- Telefon: 289-17740
- Raum: 03.09.039

- Sprechstunde: Mittwoch 13-14 Uhr
  (oder nach Vereinbarung)

Hinweis

Am Montag, dem 3. Dezember 2007 entfällt die Vorlesung aufgrund einer Dienstreise.
Gleiches gilt für die Übung am Dienstag, dem 4. Dezember 2007.

Voraussetzungen

- Voraussetzungen:
  Stoff des Informatik Grundstudiums
  (Einführung in die Informatik, Diskrete Strukturen)
- vorteilhaft:
  Effiziente Algorithmen und Datenstrukturen I/II

Inhalt

- Inhalt:
  - Zentralitätsindizes
  - Dichte in (Teil-)Graphen
  - Alternative Algorithmen für Zusammenhangsprobleme
  - Clustering
  - Netzwerk-Statistik
  - Netzwerk-Vergleich
  - Spektrale Analyse
  - Robustheit

Literatur

- Die Vorlesung orientiert sich an folgendem Buch:

U. Brandes, Th. Erlebach (Eds.):
Network Analysis – Methodological Foundations

Webzugriff:
http://www.springerlink.com/openurl.asp?genre=issue&issn=0302-9743&volume=3418

(Automatische Proxy-Konfiguration: http://pac.lrz-muenchen.de/)

Übersicht

Netzwerke

- *Netzwerk*
  Objekt, bestehend aus *Elementen* und *Interaktionen* bzw. *Verbindungen* zwischen den Elementen

- Ein Netzwerk ist ein *informales Konzept*.
  Wir haben dafür keine exakte Definition.
  Die Elemente und insbesondere ihre Verbindungen können einen ganz unterschiedlichen Charakter haben.
  Manchmal manifestieren sie sich in real existierenden Dingen, machmal sind sie nur gedacht (virtuell).

Beispiele für Netzwerke

- Beispiele:
  - Kommunikationsnetze (Internet, Telefonnetz),
  - Verkehrsnetze (Straßennetz, Schienennetz, Flugnetz, Nahverkehrsnetz),
  - Versorgungsnetzwerke (Strom, Wasser, Gas, Erdöl),
  - wirtschaftliche Netzwerke (Geld- und Warenströme, Handel)
  - biologische Netzwerke (Metabolische und Interaktionsnetzwerke),
  - soziale Netzwerke (Communities),
  - Publikationsnetzwerke

Erkenntnis

- Erkenntnis:
  Netze sind allgegenwärtig im täglichen Leben jedes Menschen.
  Wenn sie nicht richtig funktionieren, kann das weitreichende Folgen haben (z.B. Stromausfall bei Überlastung).

- Folgerung:
  Es kann von großem Vorteil sein, wenn man Verfahren kennt, um Netzwerke zu analysieren, d.h. die Stärken und die Schwächen von Netzwerken festzustellen und Netzwerkvorgänge zu optimieren.

Graphen

- *Graph*
  abstraktes Objekt, bestehend aus
  - einer Menge von *Knoten* (engl. nodes, vertices) und
  - einer Menge von *Kanten* (engl. edges, lines, links), die jeweils ein Paar von Knoten verbinden.

- Notation: $G = (V, E)$

- Anzahl der Knoten: $n = |V|$
  Anzahl der Kanten: $m = |E|$

Gerichtete und ungerichtete Graphen

- Wir unterscheiden ungerichtete und gerichtete Kanten (bzw. Graphen):
  - ungerichtet: $E \subseteq \{\{v, w\} : v \in V, w \in V\}$ (ungeordnetes Paar von Knoten bzw. 2-elementige Teilmenge)
  - gerichtet: $E \subseteq \{(v, w) : v \in V, w \in V\}$, also $E \subseteq V \times V$ (geordnetes Paar von Knoten)
- Sind zwei Knoten v und w durch eine Kante e verbunden, dann nennt man
  - v und w *adjazent* bzw. *benachbart*
  - v und e *inzident* (ebenso w und e)
- Anzahl der Nachbarn eines Knotens v: *Grad* $\deg(v)$
  Bei gerichteten Graphen unterscheidet man
  - *Eingangsgrad*: $\deg^-(v) : |\{(w, v) \in E\}|$ und
  - *Ausgangsgrad*: $\deg^+(v) : |\{(v, w) \in E\}|$

Annahmen

Wir gehen meist von folgenden Annahmen aus:

- Der Graph (also die Anzahl der Knoten und Kanten) ist *endlich*.
- Der Graph ist *einfach*, d.h. E ist eine Menge und keine Multimenge (anderenfalls nennen wir G einen Multigraph).
- In den meisten Fällen gehen wir davon aus, dass der Graph keine *Schleifen* enthält (Kanten von v nach v).

Gewichtete Graphen

In Abhängigkeit von dem betrachteten Problem wird den Kanten und/oder Knoten oft ein Wert (das *Gewicht*) zugeordnet (evt. auch mehrere), z.B.

- Längen / Signallaufzeiten,
- Kosten,
- Kapazitäten / Bandbreite,
- Ähnlichkeiten,
- Verkehrsdichte, etc.

Wir nennen den Graphen dann

- *knotengewichtet* bzw.
- *kantengewichtet*

Beispiel: $w : E \mapsto \mathbb{R}$
Schreibweise: $w(e)$ für das Gewicht einer Kante $e \in E$

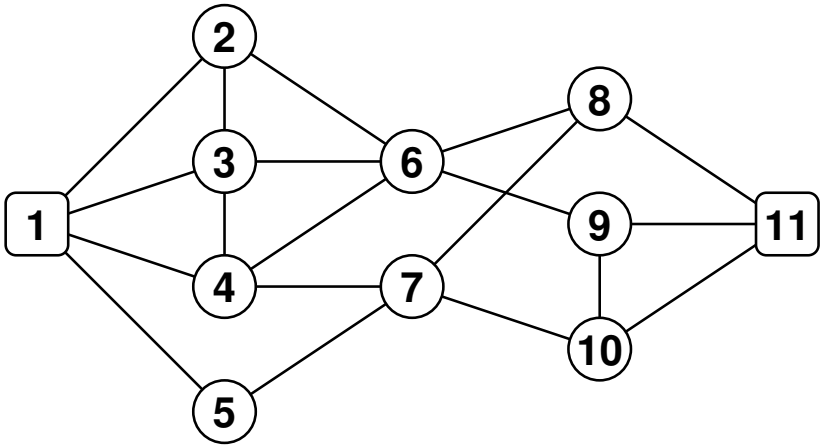
Wege, Pfade und Kreise

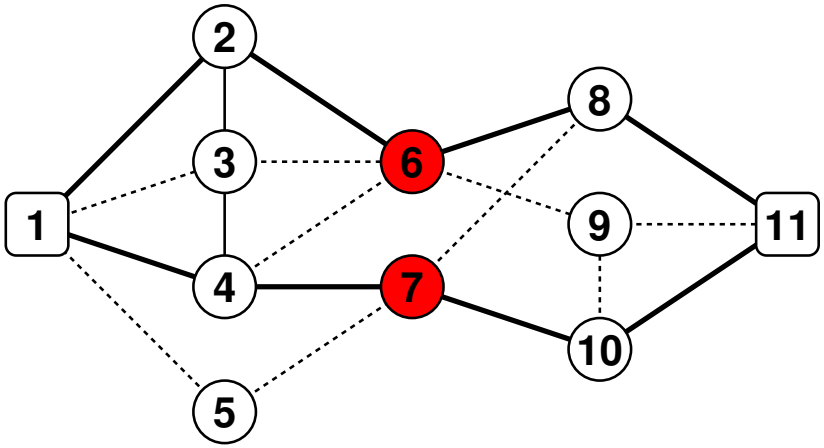
- Ein *Weg* (engl. walk) in einem Graphen $G = (V, E)$ ist eine alternierende Folge von Knoten und Kanten $x_0, e_1, \ldots, e_k, x_k$, so dass
  - $\forall i \in [0, k] : x_i \in V$ und
  - $\forall i \in [1, k] : e_i = \{x_{i-1}, x_i\}$ bzw. $e_i = (x_{i-1}, x_i) \in E$.
- Die *Länge* eines Weges ist die Anzahl der enthaltenen Kanten.

- Wir bezeichnen einen Weg als
  - *Pfad*, falls $e_i \neq e_j$ für $i \neq j$,
  - *einfachen Pfad*, falls $x_i \neq x_j$ für $i \neq j$.

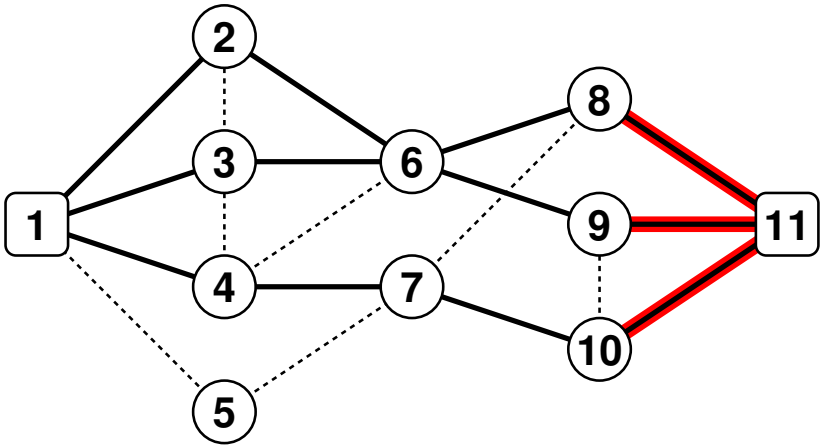
- Ein Weg heißt *Kreis*, falls $x_0 = x_k$.

Disjunkte s-t-Pfade





2 knotendisjunkte 1-11-Pfade



3 kantendisjunkte 1-11-Pfade

Graphrepräsentation

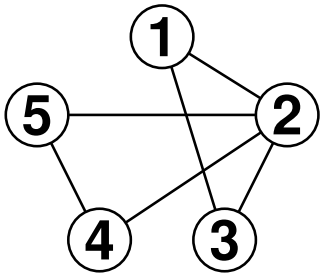
Wie kann man Graphen im Computer repräsentieren?
Vor- und Nachteile bei z.B. folgenden Fragen:

- Sind zwei gegebene Knoten v und w adjazent?
- Was sind die Nachbarn eines Knotens?
- Welche Knoten sind (direkte oder indirekte) Vorgänger bzw. Nachfolger eines Knotens v in einem gerichteten Graphen?
- Wie aufwendig ist es, einen Knoten einzufügen oder zu löschen?

Graphrepräsentationen

- Kantenliste
- Adjazenzmatrix
- Adjazenzarray
- Adjazenzliste
- Inzidenzmatrix
- implizit

Kantenliste



$$\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{2,5\},\{4,5\}$$

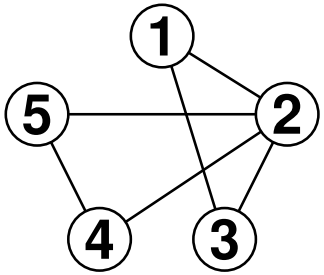
Vorteil:

- Speicherbedarf $2m + \mathcal{O}(n)$
- Einfügen und Löschen von Knoten und Kanten in $\mathcal{O}(1)$

Nachteil:

- Nachbarn nur in $\mathcal{O}(m)$ feststellbar

Adjazenzmatrix



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

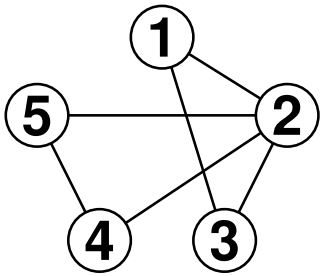
Vorteil:

- in $\mathcal{O}(1)$ feststellbar, ob zwei Knoten Nachbarn sind
- ebenso Einfügen und Löschen von Kanten

Nachteil:

- kostet $\mathcal{O}(n^2)$ Speicher, auch bei Graphen mit $o(n^2)$ Kanten
- Finden aller Nachbarn eines Knotens kostet $\mathcal{O}(n)$
- Hinzufügen neuer Knoten ist schwierig

Inzidenzmatrix

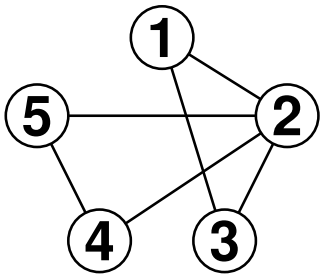


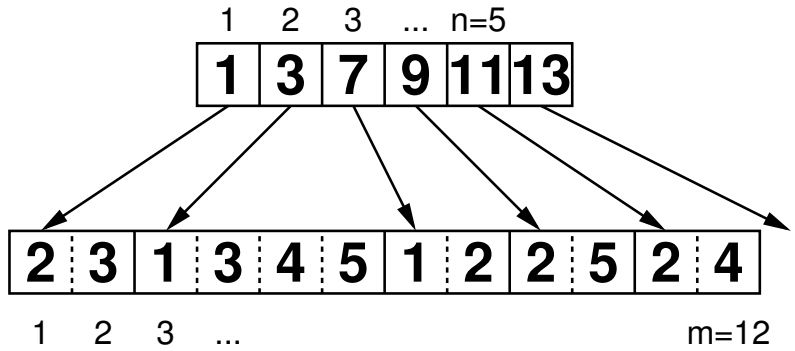
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Nachteil:

- kostet $\mathcal{O}(mn)$ Speicher

Adjazenzarray





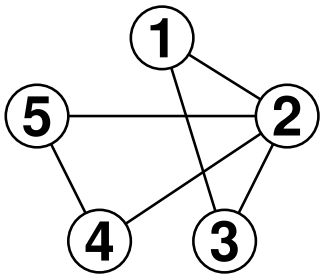
Vorteil:

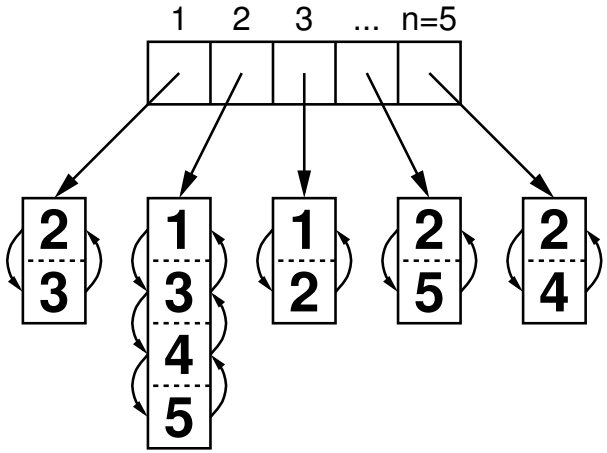
- Speicherbedarf $n + m + \Theta(1)$
  (besser Kantenliste mit $2m$)

Nachteil:

- Einfügen und Löschen von Kanten ist schwierig

Adjazenzliste





Unterschiedliche Varianten:
einfach/doppelt verkettet, linear/zirkulär

Vorteil:

- Einfügen und Löschen von Kanten in $\mathcal{O}(1)$
- mit unbounded arrays etwas cache-effizienter

Nachteil:

- Zeigerstrukturen verbrauchen relativ viel Platz und Zugriffszeit

Graphtraversierung

- Tiefensuche (depth first search, dfs)
  - Topologische Sortierung (bei DAGs)
  - Zweifachzusammenhangskomponenten
  - Starke Zusammenhangskomponenten

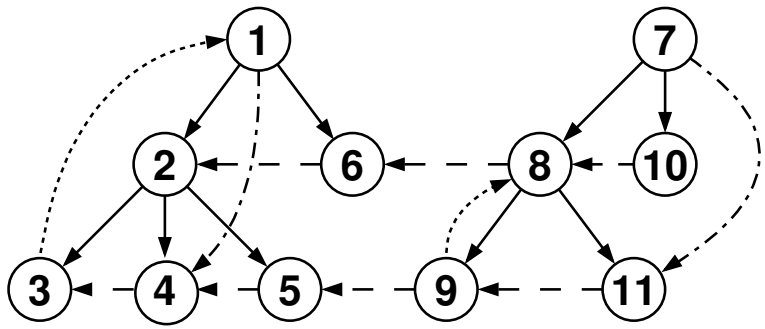
- Breitensuche (breadth first search, bfs)
  - schichtenweise Traversierung in aufsteigendem Abstand vom Ursprungsknoten

Traversierung / Kantentypen

DFS und BFS erzeugen Bäume bzw. Wälder und teilen die Kanten in folgende Klassen:

- Baumkanten (tree edges)
- Vorwärtskanten (forward edges)
- Rückwärtskanten (backward edges)
- Querkanten (cross edges)

Beispiel:



Blöcke und Artikulationsknoten

Definition

Ein Knoten v eines Graphen G heißt *Artikulationsknoten*, wenn sich die Anzahl der Zusammenhangskomponenten von G durch das Entfernen von v erhöht.

Definition

Die *Zweifachzusammenhangskomponenten* oder *Blöcke* eines Graphen sind die maximalen Teilgraphen, die 2-fach zusammenhängend sind.

Blöcke und DFS

Modifizierte DFS nach R. E. Tarjan:

- `num`$[v]$: DFS-Nummer von v
- `low`$[v]$: minimale Nummer `num`$[w]$ eines Knotens w, der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante erreicht werden kann

- `low`$[v]$: Minimum von
  - `num`$[v]$
  - `low`$[w]$, wobei w ein Kind von v im DFS-Baum ist
  - `num`$[w]$, wobei $\{v, w\}$ eine Rückwärtskante ist

Blöcke und DFS

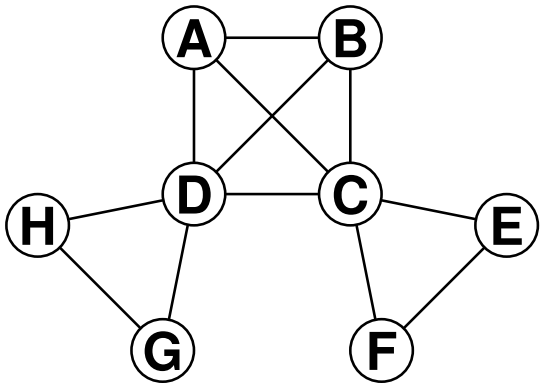
Lemma

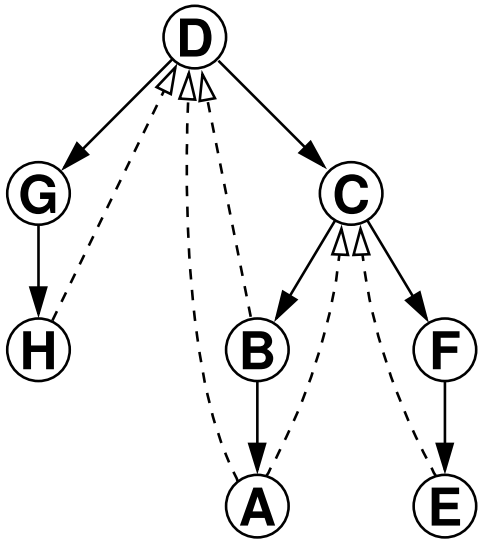
Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph und T ein DFS-Baum in G.
Ein Knoten $a \in V$ ist genau dann ein Artikulationsknoten, wenn

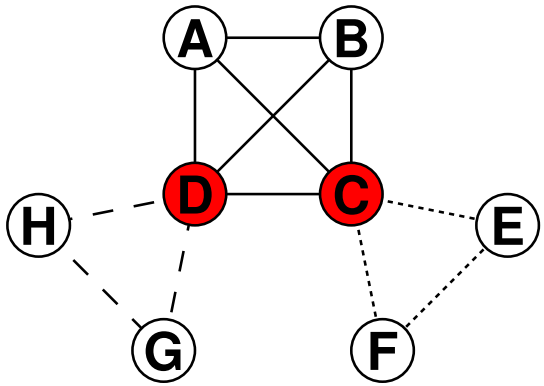
- *a die Wurzel von T ist und mindestens 2 Kinder hat, oder*
- *a nicht die Wurzel von T ist und es ein Kind b von a mit $low[b] \geq num[a]$ gibt.*

Blöcke und DFS

Die Kanten werden auf einem Stack gesammelt und nach der Erkennung eines Artikulationsknotens wird der gesamte Block abgepflückt.





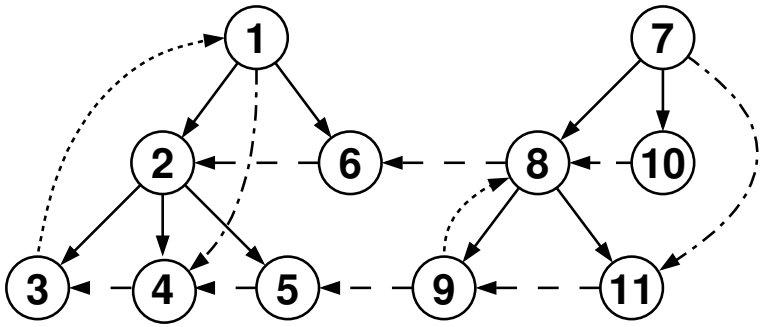


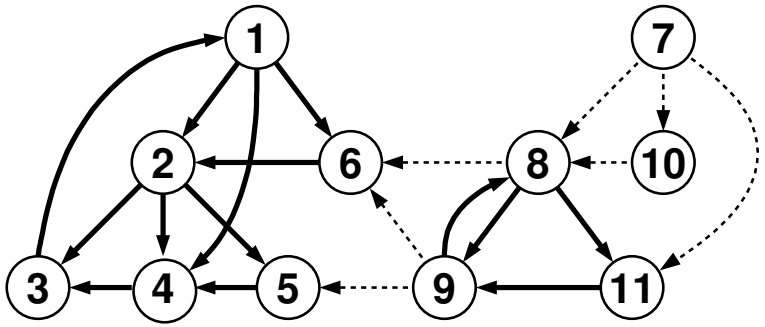
Starke Zhk. und DFS

Modifizierte DFS nach R. E. Tarjan:

- `num[`v`]`: DFS-Nummer von v
- `low[`v`]`: minimale Nummer `num[`w`]` eines Knotens w, der von v aus über beliebig viele (≥ 0) Baumkanten abwärts, evt. gefolgt von einer einzigen Rückwärtskante oder einer Querkante zu einer ZHK, deren Wurzel echter Vorfahre von v ist, erreicht werden kann
- `low[`v`]`: Minimum von
  - `num[`v`]`
  - `low[`w`]`, wobei w ein Kind von v im DFS-Baum ist
  - `num[`w`]`, wobei $\{v, w\}$ eine Rückwärtskante ist
  - `num[`w`]`, wobei $\{v, w\}$ eine Querkante ist und die Wurzel der starken Zusammenhangskomponente von w ist Vorfahre von v
- Ein Knoten v ist genau dann Wurzel einer starken Zusammenhangskomponente, wenn `num[`v`]`=`low[`v`]`.

Starke Zusammenhangskomponenten





Übersicht

Zentralitätsindizes

- Manche Knoten / Kanten in Netzwerken sind wichtiger, zentraler oder einflussreicher als andere.
- Zentralitätsmaße bzw. -indizes (kurz Zentralitäten) versuchen, diese Eigenschaften durch skalare Werte zu quantifizieren.
- Es gibt jedoch kein universelles Zentralitätsmaß, das jeder Anwendung gerecht wird.
  Das 'richtige' Maß hängt vom Kontext der Anwendung ab.

Beispiel: Leader election (1)

Bsp.: Wahl eines Klassensprechers

- Knoten entsprechen Personen
- Kante von Knoten A nach B, wenn Person A für Person B stimmt

- Person ist zentraler, je höher die Anzahl der erhaltenen Stimmen ist
  $\Rightarrow$ in-degree centrality

Beispiel: Leader election (2)

Bsp.: Wahl eines Klassensprechers

- Knoten entsprechen Personen
- Kante von Knoten A nach B, wenn Person A Person B überzeugt hat, für seinen/ihren Favoriten zustimmen
- Einfluss-Netzwerk

- Person ist zentraler, je mehr diese Person gebraucht wird, um die Meinung anderer zu transportieren
  $\Rightarrow$ betweenness centrality

Beispiel: Leader election (3)

Bsp.: Wahl eines Klassensprechers

- Knoten entsprechen Personen
- Kante von Knoten A nach B, wenn Person A mit Person B befreundet ist

- Person ist zentraler, je mehr Freunde diese Person hat und je zentraler diese Freunde sind
  $\Rightarrow$ feedback centrality

Kantenzentralität

Genau wie die Zentralität von Knoten kann man auch die Wichtigkeit von Kanten betrachten
Beispiel: Internet

- Backbone: Verbindungen zwischen den Kontinenten gibt es wenige und sie müssen eine große Kapazität haben
- Kantenzentralität
  - Beteiligung einer Kante an kürzesten Wegen usw.
    $\Rightarrow$ betweenness edge centrality
  - Veränderung von Netzwerk-Parametern durch Löschen der Kante
    $\Rightarrow$ (edge) vitality (Bsp. flow betweenness vitality)

Definition: Zentralitätsindex

Abgesehen von der Intuition

- Wichtigkeit,
- Prestige,
- Einfluss,
- Kontrolle,
- Unentbehrlichkeit

gibt es keine allgemeingültige (formale) Definition von Zentralität.

Mindestanforderung:
Das Maß darf nur von der Struktur des Graphen abhängen.

Graph-Isomorphie

Definition (Isomorphie)

Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ sind isomorph ($G_1 \simeq G_2$), falls es eine Bijektion $\phi : V_1 \to V_2$ gibt, so dass

$$(u, v) \in E_1 \quad \Leftrightarrow \quad (\phi(u), \phi(v)) \in E_2$$

Definition (Struktureller Index)

Sei $G = (V, E)$ ein gewichteter (gerichteter oder ungerichteter) graph und sei X die Knotenmenge (V) oder die Kantenmenge (E).
Eine reell-wertige Funktion s heißt *struktureller Index* genau dann, wenn die folgende Bedingung erfüllt ist:

$$\forall x \in X : \quad G \simeq H \Rightarrow s_G(x) = s_H(\phi(x))$$

Grad-Zentralität

Grad-Zentralität (degree centrality):

$$c_D(v) = \deg(v)$$

- lokales Maß,
  hängt nur von der direkten Nachbarschaft eines Knotens ab

Exzentrizität

Anwendungsbeispiel:
Positionierung eines Hospitals oder einer Feuerwehrstation

Ziel: Minimierung der maximal notwendigen Anfahrtzeit

Exzentrizität eines Knotens $v \in V$:

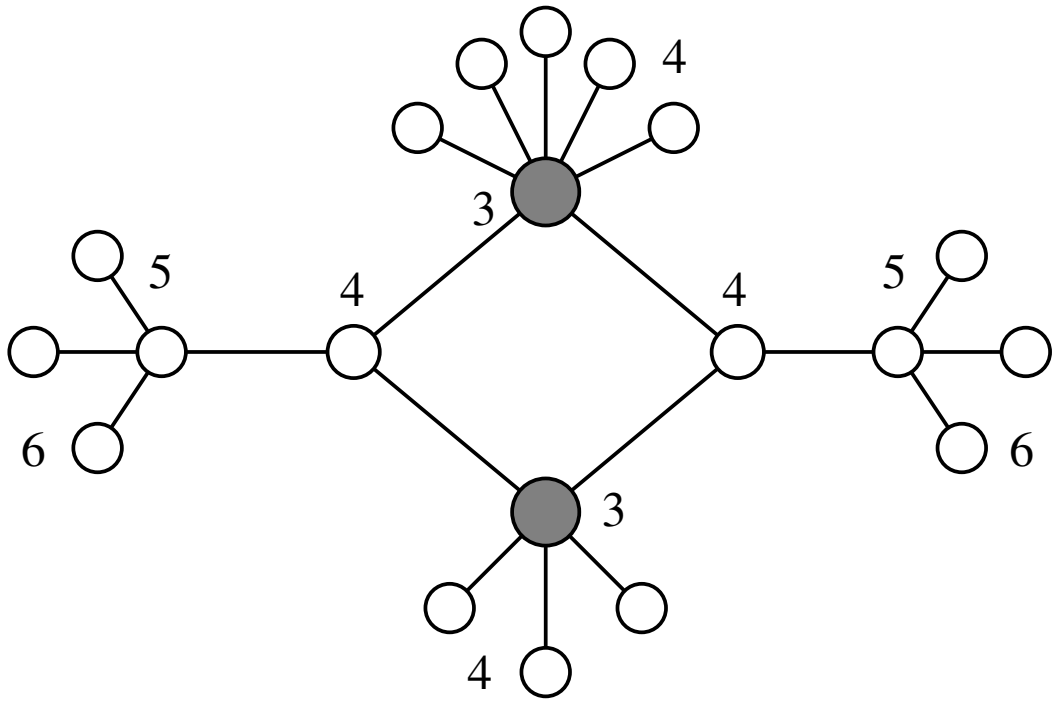
$$e(v) = \max_{w \in V} \{d(v, w)\}$$

Zentralitätsmaß:

$$c_E(v) = \frac{1}{e(v)}$$

Optimaler Standort:
Knoten v mit minimalem Wert $e(v)$ (*Zentrum* von G)

Exzentrizität / Beispiel



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Closeness

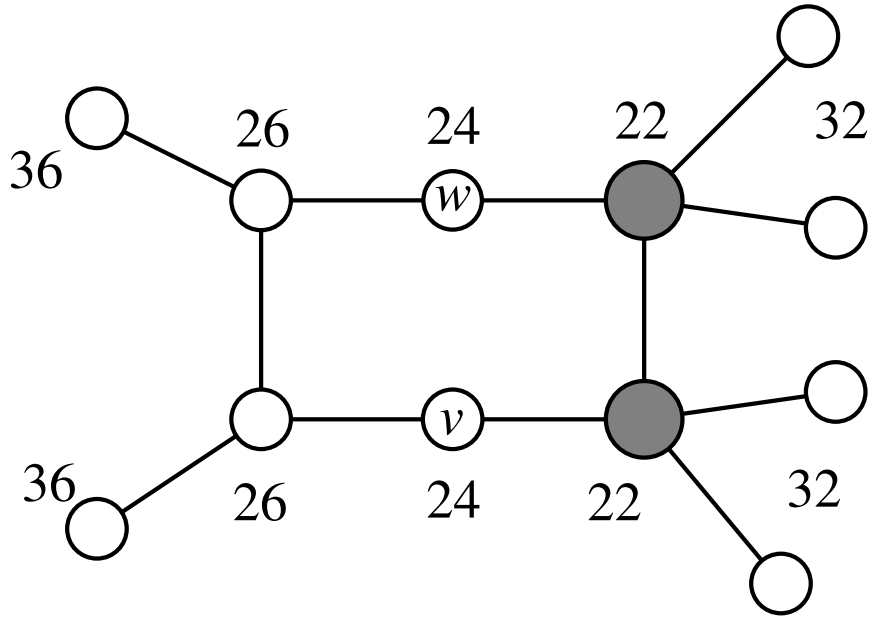
Anwendungsbeispiel:
Positionierung eines Einkaufszentrums
(minisum location / median / service facility location problem)

Ziel:
Minimierung der Summe der Entfernungen zu den anderen Knoten
(und damit auch der durchschnittlichen Entfernung)

Zentralitätsmaß:

$$c_C(v) = \frac{1}{\sum_{w \in V} d(v, w)}$$

Closeness / Beispiel



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Graue Knoten sind am wichtigsten bezüglich Closeness,
v und w sind zentraler in Hinsicht auf Exzentrizität

Radiality

ähnlich zu Closeness

$$c_R(v) = \frac{\sum_{w \in V} (\Delta_G + 1 - d(v, w))}{n - 1}$$

Δ_G: Durchmesser des Graphen
(größte Distanz zweier Knoten, nicht Länge des längsten Pfads!)

Zentroid-Wert

Konkurrenzsituation:

- Knoten repräsentieren Kunden, die beim nächstgelegenen Geschäft einkaufen
- Annahme: zwei Anbieter, der zweite Anbieter berücksichtigt bei der Standortwahl den Standort des ersten Anbieters
- Fragen:
  - Welchen Standort muss der erste Anbieter wählen, damit er durch den zweiten Anbieter möglichst wenig Kunden verliert?
  - Ist es vorteilhaft als Erster auswählen zu können?

Zentroid-Wert

geg.: ungerichteter zusammenhängender Graph G

Für ein Paar von Knoten u und v sei $\gamma_u(v)$ die Anzahl der Knoten, deren Distanz zu u kleiner ist als zu v:

$$\gamma_u(v) = |\{w \in V : d(u,w) < d(v,w)\}|$$

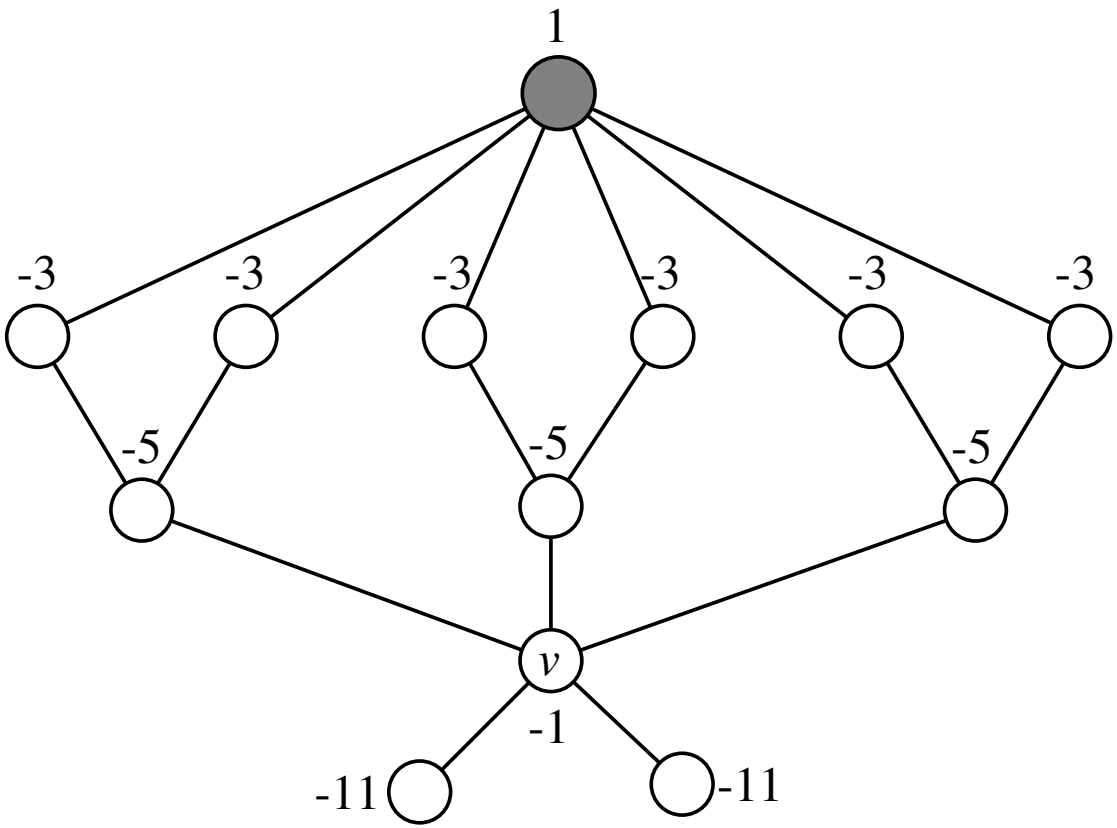
Wähle Knoten u, Gegenspieler wählt Knoten v
Resultierende Anzahl Kunden:

$$\Rightarrow \quad \gamma_u(v) + \frac{n - \gamma_u(v) - \gamma_v(u)}{2} = \frac{n + \gamma_u(v) - \gamma_v(u)}{2}$$

$\Rightarrow$ Gegenspieler minimiert $f(u,v) = \gamma_u(v) - \gamma_v(u)$
Zentralitätsmaß:

$$c_F(u) = \min_{v \in V-u} \{f(u,v)\}$$

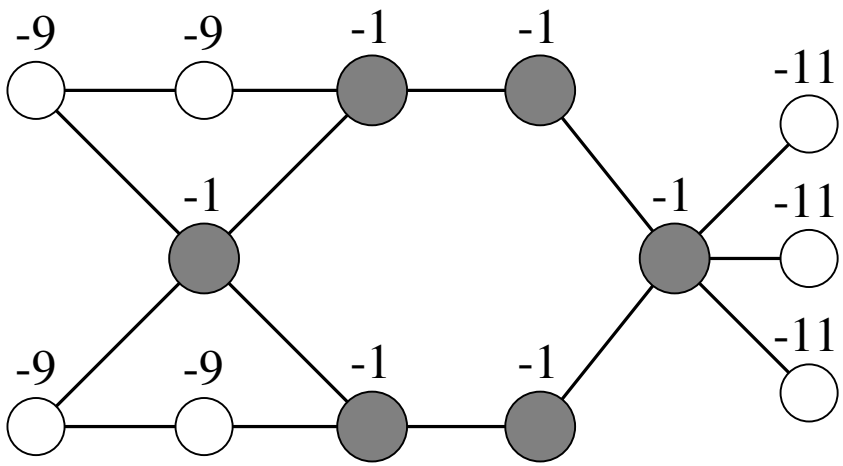
Zentroid-Wert / Beispiel 1



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Der graue Knoten hat maximalen Zentroid-Wert,
aber v ist der Knoten mit maximaler Closeness.

Zentroid-Wert / Beispiel 2



(Quelle: Brandes/Erlebach (Eds.): Network Analysis)

Die grauen Knoten haben maximalen Zentroid-Wert,
aber selbst sie haben einen negativen Wert.
$\Rightarrow$ Es ist hier vorteilhaft, erst als Zweiter zu wählen.

Zusammenfassung

- Exzentrizität, Closeness und Zentroid-Wert sind strukturelle Indizes.
- Die Knoten maximaler Zentralität unterscheiden sich bei den verschiedenen Maßen.
- Im Gegensatz zu Exzentrizität und Closeness kann der Zentroid-Wert negativ sein.

Knoten maximaler Zentralität

Definition

Für ein Zentralitätsmaß c sei die Menge der Knoten mit maximaler Zentralität in einem Graphen G definiert als

$$S_c(G) = \{v \in V : \forall w \in Vc(v) \geq c(w)\}$$

Eigenschaft des Baum-Zentrums

Satz (Jordan, 1869)

Für jeden Baum T gilt, dass sein Zentrum aus höchstens zwei Knoten besteht, die miteinander benachbart sind.

Beweis.

Baum aus höchstens 2 Knoten $\Rightarrow$ trivial
Für jeden Knoten v von T kann nur ein Blatt exzentrisch sein.
Knoten v ist dann exzentrisch zu Knoten w, wenn $d(v, w) = e(w)$
Betrachte nun den Baum T', der durch Entfernen aller Blätter aus T entsteht. Die Exzentrizität jedes Knotens in T' ist um genau Eins kleiner als in T. Deshalb haben beide Bäume das gleiche Zentrum (falls T' nicht leer ist). Die Fortsetzung dieses Verfahrens führt zwangsläufig zu einem Baum, der aus genau einem oder genau zwei adjazenten Knoten besteht. □

Berechnung des Baum-Zentrums

Der vorangegangene Beweis impliziert einen einfachen Algorithmus für die Berechnung des Zentrums eines Baums, der nicht einmal die Berechnung der Exzentrizität der einzelnen Knoten erfordert.

Eigenschaft des Graph-Zentrums

Satz

Sei G ein zusammenhängender ungerichteter Graph.
Dann existiert ein Block (zweifach zusammenhängender Teilgraph oder Brücke oder Artikulationsknoten) in G, der alle Knoten des Zentrums $\mathcal{C}(G)$ enthält.

Oder anders formuliert:
Die Knoten des Graph-Zentrums befinden sich alle in einem Block.

Eigenschaft des Graph-Zentrums

Beweis.

- Annahme: Es gibt keine Zweifachzusammenhangskomponente in G, die alle Knoten des Zentrums $\mathcal{C}(G)$ enthält.

$\Rightarrow$ G enthält einen Artikulationsknoten u, so dass G in nicht verbundene Teilgraphen G_1 und G_2 zerfällt, die jeweils mindestens einen Knoten aus $\mathcal{C}(G)$ enthalten (oder $|\mathcal{C}(G)| < 3 \ldots$).

- Sei v ein exzentrischer Knoten von u und P ein entsprechender kürzester Pfad (der Länge $e(u)$) zwischen u und v. O.B.d.A. sei v aus G_2.

$\Rightarrow$ $\exists$ Knoten $w \in \mathcal{C}(G)$ in G_1, der nicht zu P gehört

$\Rightarrow$ $e(w) \geq d(w,u) + d(u,v) \geq 1 + e(u)$

$\Rightarrow$ wegen $e(w) > e(u)$ gehört w nicht zu $\mathcal{C}(G)$ (Widerspruch)

Graph-Median

$$s(G) = \min_{v \in V} \left\{ \sum_{w \in V} d(v, w) \right\}$$

Median:

$$\mathcal{M}(G) = \left\{ v \in V : \sum_{w \in V} d(v, w) = s(G) \right\}$$

Eigenschaften des Graph-Medians

Satz

Sei G ein zusammenhängender ungerichteter Graph.
Dann existiert ein zweifach zusammenhängender Teilgraph in G, der alle Knoten des Medians $\mathcal{M}(G)$ enthält.
Oder anders formuliert:
Die Knoten des Graph-Medians befinden sich alle in einer Zweifachzusammenhangskomponente (einem Block).

Beweis.

(analog zu Beweis für Graph-Zentrum) □

Folgerung

Der Median eines Baums besteht entweder aus einem einzelnen Knoten oder aus zwei adjazenten Knoten.

Graph-Zentroid

$$f(G) = \max_{v \in V} \{c_F(v)\}$$

Zentroid:

$$\mathcal{Z}(G) = \{v \in V : \ c_F(v) = f(G)\}$$

Eigenschaften des Graph-Zentroids

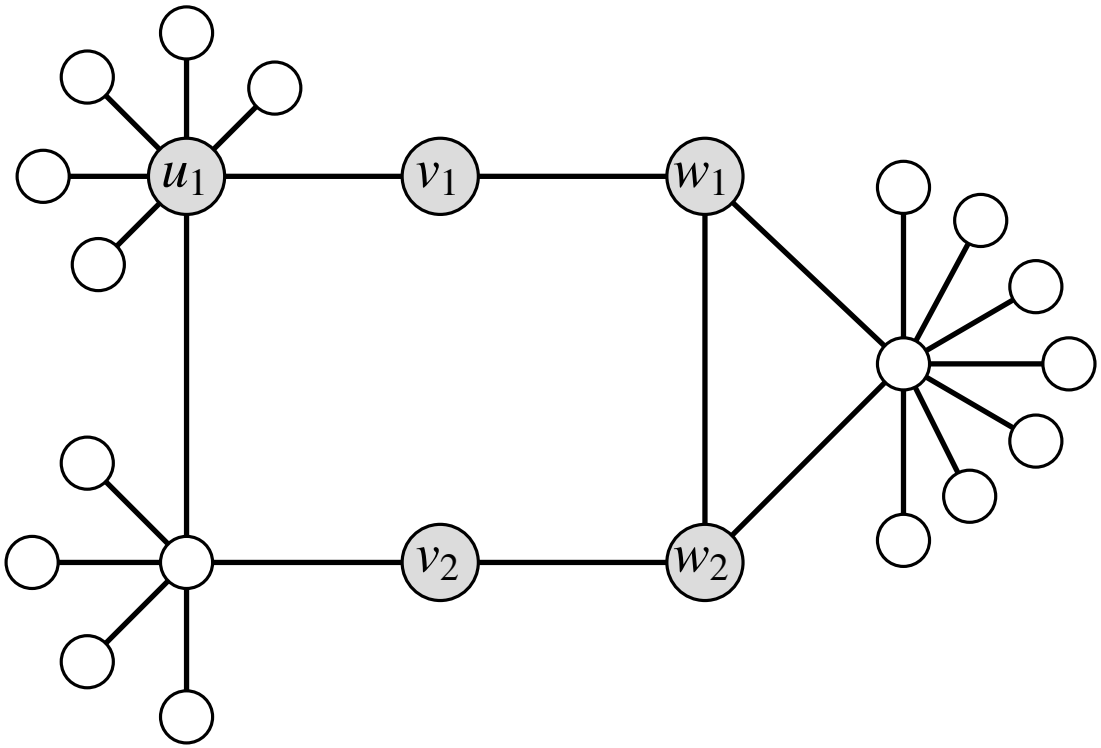
Satz (Slater)

Für jeden Baum sind Median und Zentroid identisch.

Satz (Smart & Slater)

In jedem zusammenhängenden ungerichteten Graphen liegen Median und Zentroid in der gleichen Zweifachzusammenhangskomponente.

Beispiel



$\mathcal{C}(G) = \{v_1, v_2\}$, $\mathcal{M}(G) = \{u_1\}$, $\mathcal{Z}(G) = \{w_1, w_2\}$

Unterschiedlichkeit der Maße

Satz

Für drei beliebige zusammenhängende ungerichtete Graphen H_1, H_2 und H_3 und eine beliebige natürliche Zahl $k \geq 4$ existiert ein ungerichteter zusammenhängender Graph G, so dass

- $G[\mathcal{C}(G)] \simeq H_1$,
- $G[\mathcal{M}(G)] \simeq H_2$,
- $G[\mathcal{Z}(G)] \simeq H_3$, *und*
- *die Distanz zwischen je zwei der zentralen Mengen ist mindestens k.*

Stress-Zentralität

- Anzahl kürzester Pfade zwischen Knoten s und t, die $v \in V$ bzw. $e \in E$ enthalten: $\sigma_{st}(v)$
- Stress-Zentralität:

$$c_S(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \sigma_{st}(v)$$

$$c_S(e) = \sum_{s \in V} \sum_{t \in V} \sigma_{st}(e)$$

- Intuition: Kommunikationsfluss durch Knoten bzw. Kanten auf (allen) kürzesten Wegen zwischen allen Knotenpaaren

Knoten- und Kanten-Stresszentralität

Lemma

In einem gerichteten Graphen $G = (V, E)$ sind Knoten- und Kanten-Stresszentralität wie folgt miteinander verknüpft:

$$c_S(v) = \frac{1}{2} \sum_{e \in \Gamma(v)} c_S(e) - \sum_{v \neq s \in V} \sigma_{sv} - \sum_{v \neq t \in V} \sigma_{vt}$$

Knoten- und Kanten-Stresszentralität

Beweis.

Betrachte einen kürzesten Pfad zwischen $s \neq t \in V$.
$\Rightarrow$ trägt genau 1 zum Stress jedes Knotens und jeder Kante bei.
Wenn man über den Beitrag dieses Pfads über alle (1 oder 2) inzidenten Kanten zu einem Knoten summiert, erhält man

- den doppelten Beitrag (2) zum Knoten selbst, falls der Pfad nicht an dem Knoten anfängt oder endet ($v \in V \setminus \{s, t\}$) und
- den einfachen Beitrag (1) zum Knoten selbst, falls der Pfad an dem Knoten anfängt oder endet ($v \in \{s, t\}$)

Die Größe

$$\sum_{v \neq s \in V} \sigma_{sv} + \sum_{v \neq t \in V} \sigma_{vt}$$

gibt an, wie oft der Knoten v der Anfangs- bzw. Endknoten eines kürzesten Pfades ist. □

Shortest-Path Betweenness

- eine Art relative Stress-Zentralität
- Anzahl kürzester Pfade zwischen $s, t \in V$: σ_{st}
- Anteil der kürzesten Wege zwischen s und t, die v enthalten:

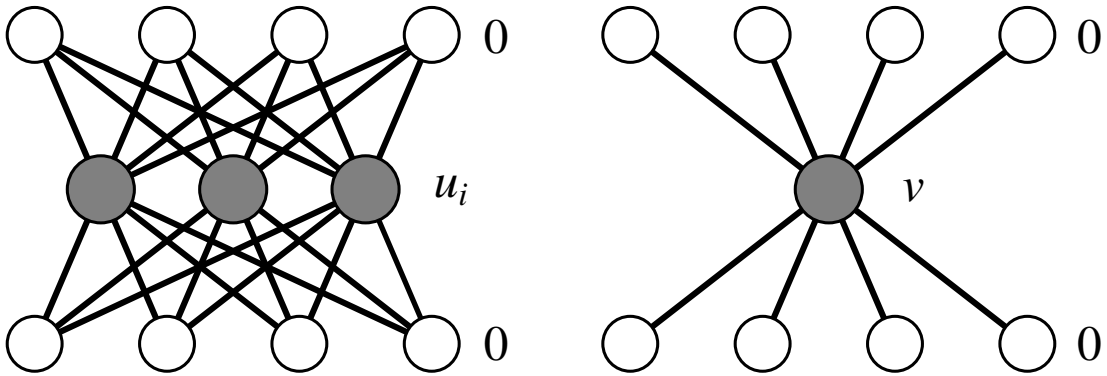
$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

$\Rightarrow$ interpretiert als Anteil oder Wahrscheinlichkeit der Kommunikation

- Betweenness-Zentralität:

$$c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v)$$

Stress und Betweenness



- Vorteil gegenüber Closeness:
  funktioniert auch bei nicht zusammenhängenden Graphen
- Normierung: Teilen durch Anzahl Paare $((n-1)(n-2))$

Betweenness für Kanten

- Anteil der Kante e am Informationsfluss (auf allen kürzesten Pfaden) zwischen den Knoten s und t

$$\delta_{st}(e) = \frac{\sigma_{st}(e)}{\sigma_{st}}$$

- Betweenness-Zentralität der Kante e:

$$c_B(e) = \sum_{s \in V} \sum_{t \in V} \delta_{st}(e)$$

Knoten- und Kanten-Betweenness-Zentralität

Lemma

In einem gerichteten Graphen $G = (V, E)$ sind Knoten- und Kanten-Betweenness-Zentralität (für kürzeste Pfade) wie folgt miteinander verknüpft:

$$c_B(v) = \sum_{e \in \Gamma^+(v)} c_B(e) - (n-1) = \sum_{e \in \Gamma^-(v)} c_B(e) - (n-1)$$

Kantengraph

Definition

Der Kantengraph des Graphen $G = (V, E)$ ist definiert als $G' = (E, K)$, wobei K die Menge der Kanten $e = ((x, y), (y, z))$ mit $(x, y) \in E$ und $(y, z) \in E$ ist.

Zwei Knoten im Kantengraph sind also benachbart, wenn die entsprechenden Kanten im ursprünglichen Graphen einen Knoten gemeinsam haben (im gerichteten Fall als Zielknoten der einen Kante und Startknoten der anderen).

$\Rightarrow$ Wende Knotenzentralität auf den Kantengraph an

Nachteile:

- Größe des Kantengraphen kann quadratisch in der Größe des Graphen sein,
- keine natürliche Interpretation / Generalisierung

Inzidenzgraph

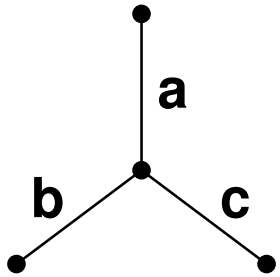
Definition

Der Inzidenzgraph des Graphen $G = (V, E)$ ist definiert als $G'' = (V'', E'')$, wobei $V'' = V \cup E$ und $E'' = \{(v, e) | \exists w : e = (v, w) \in E\} \cup \{(e, w) | \exists v : e = (v, w) \in E\})$.

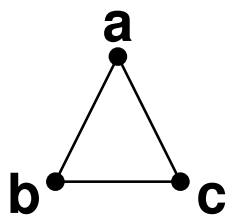
Im Inzidenzgraphen sind also ein 'echter Knoten' und ein 'Kantenknoten' benachbart, wenn der entsprechende Knoten und die entsprechende Kante im ursprünglichen Graphen inzident sind.

$\Rightarrow$ Wende Knotenzentralität auf den Inzidenzgraph an, wobei nur die Pfade zwischen 'echten Knoten' als relevant betrachtet werden

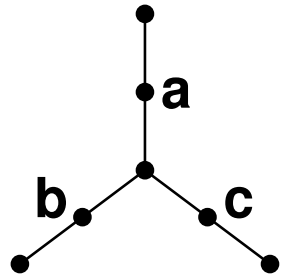
Abgeleitete Kantenzentralitäten



Graph



Kantengraph



Inzidenzgraph

Vitalität

Vitalitätsmaße bewerten die Wichtigkeit eines Knotens oder einer Kante anhand eines Qualitätsverlusts durch das Löschen des Knotens bzw. der Kante.

Definition (Vitalitätsindex)

Sei $\mathcal{G}$ die Menge aller einfachen ungerichteten ungewichteten Graphen $G = (V, E)$ und $f : \mathcal{G} \to \mathbb{R}$ eine reellwertige Funktion auf $G \in \mathcal{G}$.
Dann ist ein Vitalitätsindex $\mathcal{V}(G, x)$ definiert als die Differenz der Werte von f auf G und $G \setminus \{x\}$:

$$\mathcal{V}(G, x) = f(G) - f(G \setminus \{x\})$$

Flow Betweenness Vitality

- ähnlich zu Shortest Path Betweenness
- Motivation: Information in einem Kommunikationsnetzwerk muss sich nicht unbedingt auf kürzesten Pfaden bewegen.
- Maß: Abhängigkeit des maximalen Flusses zwischen zwei Knoten von der Existenz des betrachteten Knotens

Flow Betweenness Vitality

- f_{st}: Maximum-Fluss zwischen Knoten s und t unter Berücksichtigung der Kantenkapazitäten
- $f_{st}(v)$: Fluss zwischen Knoten s und t, der bei Maximum-Fluss von s nach t durch v gehen muss
- $\tilde{f}_{st}(v)$: Maximum-Fluss von s nach t in $G - v$

$$c_{mf}(v) = \sum_{\substack{s,t \in V \\ v \neq s, v \neq t \\ f_{st} > 0}} \frac{f_{st}(v)}{f_{st}} = \sum_{\substack{s,t \in V \\ v \neq s, v \neq t \\ f_{st} > 0}} \frac{f_{st} - \tilde{f}_{st}(v)}{f_{st}}$$

Closeness Vitality

- Wiener Index:

$$I_W(G) = \sum_{v \in V} \sum_{w \in V} d(v, w)$$

- oder in Abhängigkeit der Closeness-Zentralitäten:

$$I_W(G) = \sum_{v \in V} \frac{1}{c_C(v)}$$

- Closeness Vitality für Knoten/Kante x:

$$c_{CV}(x) = I_W(G) - I_W(G - x)$$

- Interpretation: Um wieviel steigen die Gesamtkommunikationskosten, wenn jeder Knoten mit jedem kommuniziert?

Closeness Vitality - Durchschnitt

- Durchschnittliche Distanz:

$$\overline{d}(G) = \frac{l_W(G)}{n(n-1)}$$

- Das Vitalitätsmaß auf der Basis $f(G) = \overline{d}(G)$ misst die durchschnittliche Verschlechterung der Entfernung zwischen zwei Knoten beim Entfernen eines Knotens / einer Kante x.

- Vorsicht! Beim Entfernen eines Artikulationsknotens (cut vertex) oder einer Brücke (cut edge) x ist $c_{CV}(x) = -\infty$.

Shortcut-Werte

- kein echter Vitalitätsindex im Sinne der Definition
- maximale Erhöhung eines Distanzwerts, wenn Kante $e = (v, w)$ entfernt wird

$\Rightarrow$ nur relevant für Knoten, bei denen alle kürzesten Pfade über e laufen

- maximale Erhöhung tritt direkt zwischen Knoten v und w auf
- alternativ: maximale relative Erhöhung
- Berechnung: mit $m = |E|$ Single Source Shortest Path Instanzen (und Vergleich mit der jeweiligen Kante)
- später: mit $n = |V|$ SSSP-Bäumen
- Anwendung auch auf Knotenlöschungen möglich

Stress-Zentralität als Vitalitätsindex

- Stress-Zentralität zählt die Anzahl der kürzesten Pfade, an denen ein Knoten oder eine Kante beteiligt ist

$\Rightarrow$ kann als Vitalitätsmaß betrachtet werden

- Aber: Anzahl der kürzesten Pfade kann sich durch Löschung erhöhen (wenn sich die Distanz zwischen zwei Knoten erhöht)

$\Rightarrow$ Längere kürzeste Pfade müssen ignoriert werden

Stress-Zentralität als Vitalitätsindex

- $f(G \setminus \{v\})$ muss ersetzt werden durch

$$\sum_{s \in V} \sum_{t \in V} \sigma_{st}[d_G(s,t) = d_{G \setminus \{v\}}(s,t)]$$

- Ausdruck in Klammern ist 1 (wahr) oder 0 (falsch)
- $f(G \setminus \{e\})$ analog:

$$\sum_{s \in V} \sum_{t \in V} \sigma_{st}[d_G(s,t) = d_{G \setminus \{e\}}(s,t)]$$

- Also: $f(G) - f(G - x) = \sum_{s \in V} \sum_{t \in V} \sigma_{st}(x)$
- kein echter Vitalitätsindex im Sinne der Definition, weil der Index-Wert nach der Löschung von der Distanz vor der Löschung abhängt

Elektrische Netzwerke

Elektrisches Netzwerk

- einfacher ungerichteter zusammenhängender Graph $G = (V, E)$
- Leitwertfunktion $c : E \to \mathbb{R}$
- Einspeisung $b : V \to \mathbb{R}$
  - positive Werte: eingehender Strom
  - negative Werte: ausgehender Strom
- Bedingung

$$\sum_{v \in V} b(v) = 0$$

- Kanten erhalten beliebige Orientierung $\vec{E}$

Elektrischer Fluss

Definition (Kirchhoffsche Gesetze)

Eine Funktion $x : E \to \mathbb{R}$ heißt (elektrischer) Strom falls

$$\forall v \in V : \quad \sum_{(v,w)\in\vec{E}} x(v,w) - \sum_{(w,v)\in\vec{E}} x(w,v) = b(v)$$

und

$$\sum_{e\in C} x(\vec{e}) = 0$$

für alle Zyklen (Kreise) C des Graphen G
Negative Werte bedeuten Fluss entgegengesetzt zur Kantenrichtung.

Elektrisches Potential

Definition

Eine Funktion $p: V \to \mathbb{R}$ heißt (elektrisches) Potential falls

$$\forall (v,w) \in \vec{E}: \quad p(v) - p(w) = \frac{x(v,w)}{c(v,w)}$$

- elektrisches Netzwerk $N = (G, c)$ hat einen eindeutig bestimmten Strom für jede Einspeisung b,
- ebenso Potential (bis auf einen additiven Term), also eigentlich Potentialdifferenzen (Spannung)

Laplacian Matrix

- $L = L(N)$
- $$L_{vw} = \begin{cases} \sum_{e \ni v} c(e) & \text{if } v = w \\ -c(e) & \text{if } e = \{v, w\} \\ 0 & \text{sonst} \end{cases}$$
- Für gegebenes Netzwerk $N = (G, c)$ und Einspeisung b kann man ein Potential durch Lösen der Gleichung $Lp = b$ finden.
- unit s-t-supply b_{st}:
  $b_{st}(s) = 1$, $b_{st}(t) = -1$, $\forall v \in V \setminus \{s, t\} : b_{st}(v) = 0$

Current-Flow Betweenness Centrality

- Durchsatz eines Knotens v bezüglich einer s-t-Einheitsspeisung b_{st}:

$$\tau_{st}(v) = \frac{1}{2}\left(-|b_{st}(v)| + \sum_{e \ni v} |x(\vec{e})|\right)$$

- $-|b_{st}(v)|$: Durchsatz für Anschlussknoten auf Null
- $\frac{1}{2}$ weil ein- und ausgehender Strom aufsummiert wird
- Current Flow Betweenness:

$$c_{CB}(v) = \frac{1}{(n-1)(n-2)} \sum_{s,t \in V} \tau_{st}(v)$$

Current-Flow Closeness Centrality

- Current-Flow Closeness:

$$c_{CC}(v) = \frac{n-1}{\sum_{t \neq v} p_{vt}(v) - p_{vt}(t)}$$

- Brandes / Fleischer:
  Current-Flow Closeness = Informationszentralität
- Informationszentralität:

$$c_I(v)^{-1} = nM_{vv} + \text{trace}(M) - \frac{2}{n},$$

wobei $M = (L + U)^{-1}$, L ist die Laplacian und U ist gleichgroße Matrix mit Einsen
(trace(M): Summe der Diagonalelemente)

Zufallsprozesse

- Was tun, wenn man kürzeste Pfade nicht berechnen kann? (z.B. weil man nicht das ganze Netzwerk kennt, sondern nur einen lokalen Ausschnitt)

$\Rightarrow$ zufällig einen Nachbarn des aktuellen Knotens explorieren (Random Walk)

- Beispiel: Weitergabe von Banknoten

Random Walks und Gradzentralität

Satz

In ungerichteten Graphen sind die Wahrscheinlichkeiten der stationären Verteilung bei einem kanonischen Random Walk proportionl zum Knotengrad.

$$p_{ij} = \frac{a_{ij}}{\deg(i)} \quad \Rightarrow \quad \pi_i = \frac{\deg(i)}{\sum_{v \in V} \deg(v)} = \frac{\deg(i)}{2m}$$

Random Walks und Gradzentralität

Beweis.

$$
\begin{aligned}
(\pi P)_j &= \sum_{i \in V} \pi_i p_{ij} = \frac{\sum_{i \in V} \deg(i) p_{ij}}{\sum_{v \in V} \deg(v)} \\
&= \frac{\sum_{i \in V} a_{ij}}{\sum_{v \in V} \deg(v)} = \frac{d(j)}{\sum_{v \in V} \deg(v)} = \pi_j
\end{aligned}
$$

□

π bzw. π_i: Wahrscheinlichkeit(en) der stationären Verteilung
P bzw. p_{ij}: Übergangswahrscheinlichkeit(en) von Knoten i nach j
a_{ij}: Eintrag (i, j) der Adjazenzmatrix

Random Walk Betweenness Centrality

- Knoten s will Knoten t eine Nachricht schicken, kennt aber nicht den kürzesten Weg

$\Rightarrow$ Random Walk

- Falls die Nachricht bei t ankommt, wird sie absorbiert
- M bzw. m_{ij}: Wahrscheinlichkeit, dass Knoten j die Nachricht an Knoten i weiterschickt

$$m_{ij} = \begin{cases} \frac{a_{ij}}{\deg(j)} & \text{falls } j \neq t \\ 0 & \text{sonst} \end{cases}$$

Random Walk Betweenness Centrality

- D: Grad-Matrix

$$d_{ij} = \begin{cases} \deg(i) & \text{falls } i = j \\ 0 & \text{sonst} \end{cases}$$

- D^{-1}: Inverse von D mit Reziproken Knotengraden auf der Hauptdiagonale (sonst Null)
- Durch das Absorptionsverhalten ist die folgende Darstellung nicht ganz korrekt

$$M = A \cdot D^{-1}$$

$\Rightarrow$ Löschen von Zeile t und Spalte t:

$$M_t = A_t \cdot D_t^{-1}$$

(Index t gibt die gelöschte Zeile/Spalte an)

Random Walk Betweenness Centrality

- Alle möglichen Pfade betrachten, zusammen mit der jeweiligen Wahrscheinlichkeit, dass dieser Pfad ausgewählt wird
- Wieviele verschiedene Pfade der Länge r von Knoten i nach Knoten j gibt es?

$$(A^r)_{ij}$$

- Wahrscheinlichkeit, dass ein in s gestarteter Random Walk sich nach r Schritten bei Knoten j befindet:

$$(M_t^r)_{js}$$

- Wahrscheinlichkeit, dass sich in Schritt $r+1$ der Random Walk zu Knoten i bewegt:

$$\left(M_t^{r+1}\right)_{js} = m_{ij}^{-1} \left(M_t^r\right)_{js}$$

Random Walk Betweenness Centrality

- Wahrscheinlichkeit, dass Knoten j eine in s gestartete Nachricht zu Knoten i schickt, summiert über alle Pfade der Länge 0 bis ∞:

$$\sum_{r=0}^{\infty} m_{ij}^{-1} \left(M_t^r\right)_{js} = m_{ij}^{-1} \left[\left(I_{n-1} - M_t\right)^{-1}\right]_{js}$$

  I_{n-1} ist die Identitätsmatrix der Dimension $n-1$.
- Alle Einträge in M_t^r sind Wert zwischen 0 und 1, die Summe ist also konvergent
  (werden wir später bei den Feedback-Zentralitäten sehen).

Random Walk Betweenness Centrality

- Sei $\mathbf{s}$ ein Vektor der Dimension $n-1$, der 1 ist bei Knoten s, und sonst 0.
- Dann kann man die letzte Gleichung wie folgt umschreiben:

$$\begin{aligned} \mathbf{v}^{\mathbf{st}} &= D_t^{-1} \cdot (I_{n-1} - M_t)^{-1} \cdot \mathbf{s} \\ &= (D_t - A_t)^{-1} \cdot \mathbf{s} \end{aligned}$$

- Vektor $\mathbf{v}^{\mathbf{st}}$ beschreibt die Wahrscheinlichkeit, die Nachricht bei Knoten i zu finden, während sie auf dem Weg von s nach t ist.

Random Walk Betweenness Centrality

- Einige Random Walks haben redundante Teile, weil sie zu einem Knoten zurückkehren, bei dem sie zuvor schon gewesen sind.
- Netzwerk ist ungerichtet

$\Rightarrow$ Kreise können in beiden Richtungen durchlaufen werden und löschen sich aus

- $\mathbf{v}^{\mathbf{st}}$ ignoriert diese Kreise

Random Walk Betweenness Centrality

$\Rightarrow$ Analogie zu Strom-Fluss in elektrischen Netzwerken

- el. Netzwerk $N = (G, c)$ mit einheitlichen Kantengewichten $c(e) = 1 \quad (\forall e \in E)$
- Laplacian $L(N) = D - A$ (D: Grad-Matrix, A: Adjazenzmatrix)

$\Rightarrow$ Ein Potential p_{st} in N für ein 'unit s-t-supply' b_{st} ist eine Lösung für das System $Lp_{st} = b_{st}$.

- Matrix L hat nicht vollen Rang $\Rightarrow$ ein Potential festlegen (Knoten v, Potentiale sind eindeutig bis auf additiven Term)

$\Rightarrow$ Matrizen L_v, D_v und A_v (Zeilen und Spalten zu v gelöscht)

- L_v hat vollen Rang $\Rightarrow$ invertierbar

$\Rightarrow$ $p_{st} = L_v^{-1} b_{st} = (D_v - A_v)^{-1} b_{st}$

- entspricht $\mathbf{v^{st}} = (D_t - A_t)^{-1} \cdot \mathbf{s}$

$\Rightarrow$ Analogie zwischen RW Betw. und Current-Flow Betw.: $c_{RW\ B}(v) = c_{C\ B}(v)$

Random Walk Closeness Centrality

- gleicher Ansatz liefert Random Walk Closeness
- Mean First Passage Time (MFPT) m_{st}:
  erwartete Anzahl Knoten, die vom Start in s bis zur (ersten) Ankunft in t besucht werden

$$m_{st} = \sum_{n=1}^{\infty} n \cdot f_{st}^{(n)}$$

  $f_{st}^{(n)}$: Wahrscheinlichkeit, dass man nach genau n Schritten erstmals bei t ankommt
- $M = (I - EZ_{dg})D \ldots$

Markov-Zentralität

- Markov-Zentralität:
$$c_M(v) = \frac{n}{\sum_{s \in V} m_{sv}}$$
- sinnvoll gerichtete und ungerichtete Graphen
in gerichteten: durchschnittliche MFPT für Random Walks, die in v beginnen bzw. enden
- erwartete Anzahl von Schritten von v zu den anderen Knoten (oder von den anderen zu v)

$\Rightarrow$ eine Art durchschnittliche Random Walk Closeness

Rückkopplungen

- Ein Knoten ist umso zentraler, je zentraler seine Nachbarn sind. (Aber ein Knoten ist ja auch ein Nachbar seiner Nachbarn...)

- Notation: Vektoren statt Funktionen (für lineare Gleichungssysteme)

Der Status-Index von Katz

- Beispiel: indirekte Wahl (z.B. k und l wählen i, aber alle anderen wählen nur k oder l und damit indirekt i)

$\Rightarrow$ Zähle auch die indirekten Stimmen, aber unter Berücksichtigung eines Dämpfungsfaktors $\alpha > 0$
(je länger der Pfad, desto geringer der Einfluss)

- einfacher ungewichteter gerichteter Graph mit Adjazenzmatrix A
- $(A^k)_{ji}$: Anzahl Pfade von j nach i der Länge k
- Katz' Status-Index: (bei Konvergenz)

$$c_K(i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ji} \quad \text{bzw.} \quad \mathbf{c_K} = \sum_{k=1}^{\infty} \alpha^k (A^T)^k \mathbf{1}_n$$

Katz' Status - Konvergenz

Um Konvergenz zu garantieren, muss α beschränkt werden.

Satz

Sei A die Adjazenzmatrix von G, $\alpha > 0$ und λ_1 der größte Eigenwert von A. Dann gilt:

$$\lambda_1 < \frac{1}{\alpha} \quad \Leftrightarrow \quad \sum_{k=1}^{\infty} \alpha^k A^k \text{ konvergiert}$$

Katz' Status - Geschlossene Form

- Bei Konvergenz erhält man die geschlossene Form

$$\mathbf{c_K} = \sum_{k=1}^{\infty} \alpha^k (A^T)^k \mathbf{1}_n = \left((I - \alpha A^T)^{-1} \right) \mathbf{1}_n$$

bzw.

$$(I - \alpha A^T)\mathbf{c_K} = \mathbf{1}_n$$

- inhomogenes lineares Gleichungssystem,
Rückkopplung: $c_K(i)$ hängt von den anderen $c_K(j)$ mit $j \neq i$ ab

Eigenvektor-Zentralität von Bonacich

- Phillip Bonacich, 1972
- basiert auf Eigenvektoren der Adjazenzmatrix eines einfachen, ungerichteten, zusammenhängenden, ungewichteten Graphen
- 3 verschiedene Methoden, deren Werte sich nur um einen konstanten Faktor unterscheiden
  - $\mathbf{s}^a$ Faktor-Analyse
  - $\mathbf{s}^b$ Konvergenz einer unendlichen Reihe
  - $\mathbf{s}^c$ Lösen eines linearen Gleichungssystems (LGS)
- Modellierung:
  - Freundschaftsnetzwerk
  - Fähigkeit, Freunde zu finden
  - $\mathbf{s}^a \in \mathbb{R}^n$, so dass der i-te Eintrag $\mathbf{s}^a_i$ das Freundschaftspotential des Knotens i darstellt

Eigenvektor-Zentralität von Bonacich: Faktor-Analyse

- Produkt der Potentiale zweier Knoten i und j, also $\mathbf{s}_i^a \mathbf{s}_j^a$ soll möglichst nah beim Eintrag der Adjazenzmatrix a_{ij} liegen.

$\Rightarrow$ Minimiere

$$\sum_{i=1}^{n} \sum_{j=1}^{n} (\mathbf{s}_i^a \mathbf{s}_j^a - a_{ij})^2$$

Eigenvektor-Zentralität von Bonacich: unendliche Reihe

- Für gegebenes $\lambda_1 \neq 0$ definiere

$$\mathbf{s}^{b_0} = \mathbf{1}_n \quad \text{und} \quad \mathbf{s}^{b_k} = A\frac{\mathbf{s}^{b_{k-1}}}{\lambda_1} = A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

Satz

Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix und λ_1 der größte Eigenwert von A, dann konvergiert

$$\lim_{k\to\infty} A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

gegen einen Eigenvektor von A mit Eigenwert λ_1.

$$\mathbf{s}^b = \lim_{k\to\infty} \mathbf{s}^{b_k} = \lim_{k\to\infty} A^k \frac{\mathbf{s}^{b_0}}{\lambda_1^k}$$

Eigenvektor-Zentralität von Bonacich: LGS

- Berechnung eines Eigenvektors eines LGS
- Definiere Zentralität als Summe der Zentralitäten der Nachbarn:

$$\mathbf{s}_i^c = \sum_{j=1}^{n} a_{ij}\mathbf{s}_j^c \quad \text{bzw.} \quad \mathbf{s}^c = A * \mathbf{s}^c$$

- Löse $\lambda\mathbf{s} = A\mathbf{s}$
- Exakt ein Eigenvektor hat von Null verschiedene Werte mit gleichem Vorzeichen

Eigenvektor-Zentralität von Bonacich: LGS

Satz

Sei $A \in \mathbb{R}^{n \times n}$ die Adjazenzmatrix eines ungerichteten zusammenhängenden Graphen. Dann gilt:

- *Der größte Eigenwert λ_1 ist ein einfacher Eigenwert. (Ihm ist nur ein Eigenvektor zugeordnet.)*
- *Alle Einträge des Eigenvektors zu λ_1 sind ungleich Null und haben das gleiche Vorzeichen.*

(folgt aus Theorem von Perron/Frobenius über nichtnegative quadratische Matrizen)

Eigenvektor-Zentralität: Definition

Die drei Varianten unterscheiden sich nur durch konstante Faktoren
Allgemeine Definition der Eigenvektor-Zentralität

$$\mathbf{c_{EV}} = \frac{|\mathbf{s^c}|}{||\mathbf{s^c}||}$$

Weitere Feedback-Zentralitäten

- Hubbell-Index (Charles Hubbell, 1965)
  gewichteter gerichteter Graph
  Zentralität eines Knotens hängt von der gewichteten Summe seiner Nachbarn ab

- Verhandlungszentralität von Bonacich
  ungewichtete gerichtete Graphen
  Zentralität eines Knotens hoch, wenn seine Nachbarn keine anderen Optionen haben
  ($\Rightarrow$ berücksichtigt *negative* Rückkopplung)

- WebGraph-Zentralitäten
  - PageRank (nur Topologie)
  - Hubs & Authorities (HITS)
  - SALSA (konzeptuell eine Kombination der beiden anderen)

Übersicht

Kürzeste Pfade: SSSP / Dijkstra

Algorithmus 1 : Dijkstra-Algorithmus (SSSP)

Input : $G = (V, E), \quad \omega : E \to \mathbb{R}, \quad s \in V$

Output : Distanzen $d(s, v)$ zu allen $v \in V$

$P = \emptyset, T = V;$

$d(s, v) = \infty$ for all $v \in V; \quad d(s, s) = 0; \quad pred(s) = 0;$

while $P \neq V$ **do**

  $v = \text{argmin}_{v \in T}\{d(s, v)\};$

  $P := P \cup v; \quad T := T \setminus v;$

  for $w \in N(v)$ **do**

    if $d(s, w) > d(s, v) + \omega(v, w)$ **then**

      $d(s, w) := d(s, v) + \omega(v, w);$

      $pred(w) := v;$

Kürzeste Pfade: SSSP / Dijkstra

- Datenstruktur: Prioritätswarteschlange
  (z.B. Fibonacci Heap: amortisierte Komplexität $\mathcal{O}(1)$ für `insert` und `decreaseKey`, $\mathcal{O}(\log n)$ `deleteMin`)
- Komplexität:
  - $n-1$ `insert`
  - $n-1$ `deleteMin`
  - $O(m)$ `decreaseKey`

  $\Rightarrow$ $\mathcal{O}(m + n \log n)$
- aber: nur für nichtnegative Kantengewichte, sonst Bellman/Ford-Algorithmus in $\mathcal{O}(mn)$

Kürzeste Pfade: APSP / Floyd-Warshall

Algorithmus 2 : Floyd-Warshall APSP Algorithmus

Input : Graph $G = (V, E)$, edge weights $\omega : E \to R$

Output : Shortest path distances $d(u, v)$ between all $u, v \in V$

$d(u, v) = \infty, pred(u, v) = 0$ for all $u, v \in V$;
$d(v, v) = 0$ for all $v \in V$;
$d(u, v) = \omega(u, v), pred(u, v) = u$ for all $\{u, v\} \in E$;
for $v \in V$ **do**
  for $\{u, w\} \in V \times V$ **do**
    if $d(u, w) > d(u, v) + d(v, w)$ **then**
      $d(u, w) := d(u, v) + d(v, w)$;
      $pred(u, w) := pred(v, w)$;

Kürzeste Pfade: APSP / Floyd-Warshall

- Komplexität: $\mathcal{O}(n^3)$

Betweenness Centrality

$$c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Mögliche Berechnung:

1. Berechne Länge und Anzahl kürzester Pfade zwischen allen Knotenpaaren
2. Betrachte zu jedem Knoten v alle möglichen Paare s, t und berechne den Anteil kürzester Pfade durch v
   Bedingung(Bellman-Kriterium):

$$d(s,t) = d(s,v) + d(v,t)$$

Betweenness Centrality

1. Modifiziere Dijkstras Algorithmus
   - Ersetze einzelnen Vorgängerknoten durch eine Menge von Vorgängern $\text{pred}(s, v)$
   - Es gilt dann

$$\sigma_{sv} = \sum_{u \in \text{pred}(s,v)} \sigma_{su}$$

2. Im Fall $d(s, t) = d(s, v) + d(v, t)$ gilt

$$\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$$

   ansonsten $(d(s, t) < d(s, v) + d(v, t))$ ist $\sigma_{st}(v) = 0$

$\Rightarrow$ $\mathcal{O}(n^2)$ pro Knoten v (Summation über alle $s \neq v \neq t$), also insgesamt $\mathcal{O}(n^3)$

Betweenness Centrality (Brandes)

Abhängigkeit eines Paares (s, t) von v:

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Abhängigkeit eines Startknotens s von v:

$$\delta_{s*}(v) = \sum_{t \in V} \delta_{st}(v)$$

Betweenness von v:

$$\Rightarrow \qquad c_B(v) = \sum_{s \neq v \in V} \delta_{s*}(v)$$

Betweenness Centrality (Brandes)

Satz

Für die Abhängigkeit $\delta_{s*}(v)$ eines Startknotens $s \in V$ von den anderen Knoten $v \in V$ gilt:

$$\delta_{s*}(v) = \sum_{w:\ v \in pred(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{s*}(w))$$

Betweenness Centrality (Brandes)

Beweis.

- Definiere $\sigma_{st}(v, e)$: Anzahl kürzester s-t-Pfade die sowohl Knoten v als auch Kante e enthalten
- Definiere entsprechend

$$\delta_{st}(v, e) = \frac{\sigma_{st}(v, e)}{\sigma_{st}}$$

- Es ergibt sich

$$\delta_{s*}(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{t \in V} \sum_{w:\ v \in \text{pred}(s,w)} \delta_{st}(v, \{v, w\})$$

Betweenness Centrality (Brandes)

Beweis.

- Betrachte Knoten w, so dass $v \in \text{pred}(s, w)$
- σ_{sw} kürzeste Pfade von s nach w,
  davon σ_{sv} von s nach v gefolgt von Kante $\{v, w\}$
- Anteil σ_{sv}/σ_{sw} der Anzahl kürzester Pfade von s nach t über w
  benutzt auch die Kante $\{v, w\}$:

$$\delta_{st}(v, \{v, w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{falls } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} & \text{falls } t \neq w \end{cases}$$

Betweenness Centrality (Brandes)

Beweis.

Vertausche die Summationsreihenfolge:

$$
\begin{aligned}
\delta_{s*}(v) &= \sum_{t \in V} \sum_{w:\ v \in \mathrm{pred}(s,w)} \delta_{st}(v, \{v, w\}) \\
&= \sum_{w:\ v \in \mathrm{pred}(s,w)} \sum_{t \in V} \delta_{st}(v, \{v, w\}) \\
&= \sum_{w:\ v \in \mathrm{pred}(s,w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus w} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\
&= \sum_{w:\ v \in \mathrm{pred}(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{s*}(w))
\end{aligned}
$$

□

Betweenness Centrality (Brandes)

- Berechne n kürzeste-Wege-DAGs
  (einen für jeden Startknoten $s \in V$)
- Berechne nacheinander für alle $s \in V$ aus dem kürzeste-Wege-DAG von s die Abhängigkeiten $\delta_{s*}(v)$ für alle anderen Knoten $v \in V$
  Vorgehen: rückwärts, von den Blättern im kürzeste-Wege-Baum bzw. von der entferntesten Schicht im kürzeste-Wege-DAG zum Startknoten hin
- Summiere die einzelnen Abhängigkeiten
  (kann schon parallel während der Berechnung aufsummiert werden, um nicht $\mathcal{O}(n^2)$ Platz zu verbrauchen)

Betweenness Centrality (Brandes)

Satz

Die Betweenness-Zentralität $c_B(v)$ für alle Knoten $v \in V$ kann

- *für gewichtete Graphen in Zeit $\mathcal{O}(n(m + n \log n)) = \mathcal{O}(nm + n^2 \log n)$*
- *für ungewichtete Graphen in $\mathcal{O}(mn)$*

berechnet werden.
Der Algorithmus benötigt dabei nur $\mathcal{O}(n + m)$ Speicherplatz.

Bemerkung:
Die anderen auf kürzesten Wegen basierenden Zentralitäten kann man relativ einfach mit SSSP-Traversierung berechnen.

Shortcut-Werte

- Ziel: Berechnung der shortcut-Werte aller Kanten in einem gerichteten Graph
- Maximale Erhöhung der Länge eines kürzesten Pfades durch Entfernen einer Kante $e = (u, v) \in E$
- Berechnung der Distanz von u zu v in $G_e = (V, E \setminus \{e\})$ für alle $e = (u, v) \in E$ (denn die maximale Erhöhung betrifft ja die Endknoten der Kante)

- einfache Lösung: $m = |E|$ SSSP Aufrufe
- besser: nur n äquivalente Aufrufe

Shortcut-Werte

- Annahme: keine negativen Kreise
  ($\Rightarrow$ $d(i,j)$ ist definiert für alle Knotenpaare (i,j)),
  keine parallelen Kanten
- Idee: ein Aufruf für Knoten u berechnet shortcut-Werte für alle ausgehenden Kanten
- Vorgehen:
  - Fixiere einen Knoten u
  - $\alpha_i = d(u,i)$: Distanz von u nach i.
  - τ_i: zweiter Knoten der kürzesten Pfade von u zu i, falls dieser Knoten eindeutig ist.
    Ansonsten $\tau_i = \perp$ (impliziert zwei kürzeste Pfade der Länge α_i mit unterschiedlichen Anfangskanten).
  - β_i: Länge des kürzesten Pfades von u nach i, so dass τ_i nicht zweiter Knoten
    (∞ falls es keinen Pfad mehr gibt, $\beta_i = \alpha_i$ falls $\tau_i = bot$)

Shortcut-Werte

- Betrachte α_v, τ_v und β_v für einen Nachbarn v von u, also $(u, v) \in E$.
- Dann ist die shortcut-Distanz für (u, v) gleich α_v falls $\tau_v \neq v$ (also Kante (u, v) ist nicht einziger kürzester Pfad)
- Ansonsten, falls $\tau_v = v$, ist die neue Distanz β_v
- $\alpha_u = 0, \quad \tau_u = 0, \quad \beta_u = \infty$

$$\alpha_j = \min_{i:(i,j)\in E} (\alpha_i + \omega(i,j))$$

- Nachbarn bezüglich eingehender Kanten, die zu einem kürzesten Pfad gehören:

$$I_j = \{i : (i,j) \in E \text{ und } a_j = a_i + \omega(i,j)\}$$

Shortcut-Werte

$$\tau_j = \begin{cases} j & \text{if } I_j = \{u\}, \quad u \text{ ist Anfang, Kante } (u,j) \\ a & \text{if } \forall i \in I_j : \ a = \tau_i \\ & \text{(Alle Vorgänger haben erste Kante } (u,a)), \\ \perp & \text{sonst} \end{cases}$$

Im Fall $\tau_j = \perp$ gilt $\beta_j = \alpha_j$, sonst

$$\beta_j = \min \left\{ \min_{i:\ (i,j)\in E, \tau_i = \tau_j} \beta_i + \omega(i,j), \min_{i:\ (i,j)\in E, \tau_i \neq \tau_j} \alpha_i + \omega(i,j) \right\}$$

Shortcut-Werte

Betrachte den Pfad p der zu β_j führt, also ein kürzester Pfad p von u nach j, der nicht mit τ_j beginnt. Wenn für den letzten Knoten i vor j in p gilt $\tau_i = \tau_j$, dann startet der Pfad p bis zu i nicht mit τ_j, und dieser Pfad wird in β_i und damit in β_j berücksichtigt.

Wenn anderenfalls $\tau_i \neq \tau_j$ für den vorletzten Knoten i von Pfad p gilt, dann beginnt einer der kürzesten Pfade von u nach i nicht mit τ_j und die Länge von p ist $\alpha_i + \omega(i, j)$.

Shortcut-Werte

Mit den Rekursionen können die Werte α_i,τ_i und β_i effizient berechnet werden.
Im Fall positiver Gewichte hängt jeder Wert α_i nur von Werten α_j ab, die kleiner als α_i sind
$\Rightarrow$ Berechnung in monotoner Weise nach Dijkstra
Bei positiven Gewichten ist der kürzeste-Wege-DAG kreisfrei und die Werte τ_i können in der Reihenfolge einer topologischen Sortierung berechnet werden. (sonst stark zusammenhängende Komponenten kontrahieren)
Werte β_i hängen nur von Werten $\beta_j \leq \beta_i$ ab
$\Rightarrow$ Berechnung in monotoner Weise nach Dijkstra
Bei negativen Kantengewichten (aber keine negativen Kreise) Dijkstra durch Bellman-Ford Algorithmus und β_i durch Berechnung von $\beta'_i = \beta_i - \alpha_i$ ersetzen

Approximation von Zentralitätsindizes

- Obwohl die behandelten Zentralitätsindizes in polynomieller Zeit berechnet werden können, heißt das nicht unbedingt , dass die entsprechenden Algorithmen in der Praxis anwendbar sind.

- Beispiel:
  Betweenness-Zentralität für die Knoten des Web-Graphen lässt sich selbst mit dem Algorithmus von Brandes nicht in akzeptabler Zeit berechnen.

$\Rightarrow$ Möglichst wenige Traversierungen bzw. SSSP-Durchläufe auf dem Graphen

$\Rightarrow$ Näherungslösungen mit möglichst geringer Abweichung (mit hoher Wahrscheinlichkeit)

Approximation von Closeness

- Closeness:

$$c_C(v) = \frac{1}{\sum_{w \in V} d(v, w)}$$

- Approximation: wähle k andere Knoten $v_1, \ldots, v_k \in V$

$$\hat{c}_C(v) = \frac{k}{n} \frac{1}{\sum_{i=1}^{k} d(v, v_i)}$$

- Vorgehen (Eppstein / Wang):
  1. Wähle k Knoten $v_1, \ldots, v_k$ gleichverteilt zufällig
  2. Löse für jeden Knoten v_i das SSSP mit diesem Knoten als Startknoten und
  3. berechne für jeden Knoten $v \in V$ die Zentralität

$$\hat{c}_C(v) = \frac{k}{n \cdot \sum_{i=1}^{k} d(v, v_i)}$$

Hoeffdings Ungleichung

Satz (Hoeffdings Ungleichung)

Seien $X_1, \ldots, X_k$ unabhängige Zufallsvariablen mit $a_i \leq X_i \leq b_i$ und $\mu = \mathbb{E}\left[\sum_{i=1}^{k} X_i/k\right]$ der erwartete Durchschnitt. Dann gilt

$$\Pr\left\{\left|\frac{\sum_{i=1}^{k} X_i}{k} - \mu\right| \geq \xi\right\} \leq 2 \cdot e^{-2k^2\xi^2 / \sum_{i=1}^{k}(b_i - a_i)^2}$$

bzw. im Fall $a_i = a$, $b_i = b$ $(\forall i)$

$$\Pr\left\{\left|\frac{\sum_{i=1}^{k} X_i}{k} - \mu\right| \geq \xi\right\} \leq 2 \cdot e^{-2k\xi^2/(b-a)^2}$$

Anwendung der Hoeffding-Ungleichung

Setze

$$\begin{aligned} X_i &= n \cdot \frac{d(v_i, u)}{n-1} \\ \mu &= \frac{1}{c_C(v)} \\ a_i &= 0 \\ b_i &= \frac{n \cdot \mathrm{diam}(G)}{n-1} \end{aligned}$$

Anwendung der Hoeffding-Ungleichung

$$\begin{aligned}
\Pr\left\{\left|\frac{\sum_{i=1}^{k} X_i}{k} - \mu\right| \geq \xi\right\} &\leq 2 \cdot e^{-2k^2\xi^2 / \sum_{i=1}^{k}(b_i - a_i)^2} \\
&= 2 \cdot e^{-2k^2\xi^2 / \left(k\left(\frac{n \cdot \mathrm{diam}(G)}{n-1}\right)^2\right)} \\
&= 2 \cdot e^{-\Omega\left(k\xi^2 / \mathrm{diam}(G)^2\right)}
\end{aligned}$$

- Wähle $\xi = \epsilon \cdot \mathrm{diam}(G)$
- $k = \Theta\left(\frac{\log n}{\epsilon^2}\right)$ SSSP-Läufe

$\Rightarrow$ Wahrscheinlichkeit, einen Fehler größer als $\epsilon \cdot \mathrm{diam}(G)$ zu machen ist höchstens $\frac{1}{n}$ für jeden Wert

Laufzeit der Closeness-Approximation

- Komplexität eines SSSP-Laufs
  - $\mathcal{O}(m+n)$ in ungewichteten Graphen
  - $\mathcal{O}(m+n\log n)$ in gewichteten Graphen
- Komplexität von k SSSP-Läufen
  - $\mathcal{O}(k\cdot(m+n))$ in ungewichteten Graphen
  - $\mathcal{O}(k\cdot(m+n\log n)$ in gewichteten Graphen

$\Rightarrow$ Komplexität von $\Theta\left(\frac{\log n}{\epsilon^2}\right)$ SSSP-Läufen

- $\mathcal{O}\left(\frac{\log n}{\epsilon^2}\cdot(m+n)\right)$ in ungewichteten Graphen
- $\mathcal{O}\left(\frac{\log n}{\epsilon^2}\cdot(m+n\log n)\right)$ in gewichteten Graphen

Approximation von Betweenness

- gewichtete gerichtete Graphen
- wähle wieder k Knoten zufällig (gleichverteilt) aus
- Berechne für jeden Startknoten v_i die totalen Abhängigkeiten $\delta_{v_i*}(v)$ aller anderen Knoten v
- Berechne

$$\hat{c}_B(v) = \sum_{i=1}^{k} \frac{n}{k} \cdot \delta_{v_i*}(v)$$

- $\mathbb{E}[\hat{c}_B(v)] = c_B(v)$ für alle k und v

Anwendung der Hoeffding-Ungleichung

Setze

$$\begin{aligned} X_i &= n \cdot \delta_{v_i *} \\ \mu &= c_B(v) \\ a_i &= 0 \\ b_i &= n(n-2) \end{aligned}$$

$\delta_{v_i *}$ kann höchstens $n-2$ sein, und zwar wenn alle kürzesten Pfade, die von v_i ausgehen, über v laufen. Also ist X_i durch $n(n-2)$ begrenzt.

Anwendung der Hoeffding-Ungleichung

$$\begin{aligned}\Pr\left\{|\hat{c}_B(v) - c_B(v)| \geq \xi\right\} &\leq 2 \cdot e^{-2k^2\xi^2 / \sum_{i=1}^{k}(b_i - a_i)^2} \\ &= 2 \cdot e^{-2k^2\xi^2 / \left(k(n(n-2))^2\right)} \\ &= 2 \cdot e^{-2k\xi^2 / (n(n-2))^2}\end{aligned}$$

- Wähle $\xi = \epsilon \cdot n(n-2)$
- $k = \Theta\left(\frac{\log n}{\epsilon^2}\right)$ Startknoten / Läufe

$\Rightarrow$ Wahrscheinlichkeit, einen Fehler größer als $\epsilon \cdot n(n-2)$ zu machen ist höchstens $\frac{1}{n}$ für jeden Wert

Laufzeit der Betweenness-Approximation

- Komplexität eines Laufs (für $\delta_{v_i *}(v)$)
  - $\mathcal{O}(m+n)$ in ungewichteten Graphen
  - $\mathcal{O}(m+n\log n)$ in gewichteten Graphen
- Komplexität von k Läufen
  - $\mathcal{O}(k\cdot(m+n))$ in ungewichteten Graphen
  - $\mathcal{O}(k\cdot(m+n\log n)$ in gewichteten Graphen

$\Rightarrow$ Komplexität von $\Theta\left(\frac{\log n}{\epsilon^2}\right)$ Läufen

- $\mathcal{O}\left(\frac{\log n}{\epsilon^2}\cdot(m+n)\right)$ in ungewichteten Graphen
- $\mathcal{O}\left(\frac{\log n}{\epsilon^2}\cdot(m+n\log n)\right)$ in gewichteten Graphen

Ergebnis

- Gewinn: k anstatt n SSSP-artige Läufe

- Verfahren des normalisierten Durchschnitts basierend auf zufälligem Knoten-Sampling läßt sich auf viele andere Zentralitäten übertragen

Übersicht

Kohäsive Gruppen

Eigenschaften:

- Gegenseitigkeit
  Gruppenmitglieder wählen sich gegenseitig in die Gruppe und sind im graphtheoretischen Sinn benachbart
- Kompaktheit / Erreichbarkeit:
  Gruppenmitglieder sind gegenseitig gut erreichbar
  (wenn auch nicht unbedingt adjazent), insbesondere
  - auf kurzen Wegen
  - auf vielen verschiedenen Wegen
- Dichte:
  Gruppenmitglieder haben eine große Nachbarschaft innerhalb der Gruppe
- Separation:
  Gruppenmitglieder haben mit größerer Wahrscheinlichkeit Kontakt zu einem anderen Mitglied der Gruppe als zu einem Nicht-Gruppenmitglied

Lokale Dichte

- Eine Gruppeneigenschaft heißt *lokal*, wenn sie bestimmt werden kann, indem man nur den von der Gruppe induzierten Teilgraphen betrachtet.

$\Rightarrow$ Separation ist *nicht lokal*, weil hier auch die Verbindungen zu den anderen Knoten betrachtet werden

- Viele Definitionen von kohäsiven Gruppen verlangen außer einer Eigenschaft Π auch *Maximalität* (im Sinne von Nichterweiterbarkeit), d.h. die Gruppe darf nicht in einer anderen größeren Gruppe enthalten sein.
- Maximalität verletzt die Lokalitätsbedingung

$\Rightarrow$ Betrachten diese Eigenschaften ohne Maximalitätsbedingung

- Lokalität reflektiert wichtige Eigenschaft von Gruppen:
  - Invarianz unter Veränderung des Netzwerks außerhalb der Gruppe
  - Innere Robustheit und Stabilität ist eine wichtige Gruppeneigenschaft

Cliquen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph. Ein Knotenteilmenge $U \subseteq V$ heißt Clique genau dann, wenn der von U in G induzierte Teilgraph $G[U]$ ein vollständiger Graph ist.
Eine Clique U in G ist eine maximale Clique, falls es keine Clique U' mit $U \subset U'$ in G gibt.

Cliquen – ideale kohäsive Gruppen

Cliquen sind ideale kohäsive Gruppen:
(Sei U eine Clique der Kardinalität k.)

- Cliquen haben größtmögliche Dichte

$$\delta(G[U]) = \bar{d}(G[U]) = \Delta(G[U]) = k - 1$$

- Cliquen besitzen größtmögliche Kompaktheit

$$\text{diam}(G[U]) = 1$$

- Cliquen sind bestmöglich verbunden
  U ist $(k-1)$-fach knotenzusammenhängend und
  $(k-1)$-fach kantenzusammenhängend

Satz von Turán

Satz (Turán, 1941)

Sei $G = (V, E)$ ein ungerichteter Graph mit $n = |V|$ und $m = |E|$.
Falls $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, dann existiert eine Clique der Größe k in G.

Spezialfall:

Satz (Mantel, 1907)

Die maximale Anzahl von Kanten in einem dreiecksfreien Graphen ist $\left\lfloor \frac{n^2}{4} \right\rfloor$.

Da die meisten (z.B. soziale) Netzwerke eher dünn sind (also $o(n^2)$ Kanten haben), müssen sie nicht unbedingt von vornherein Cliquen einer bestimmten Größe > 2 enthalten.

Maximale Cliquen

- Graphen enthalten immer maximale Cliquen.
- Meistens sind es sogar viele.
- Sie können sich überlappen (ohne identisch zu sein).

Satz (Moon & Moser, 1965)

Jeder ungerichtete Graph G mit n Knoten hat höchstens $3^{\lceil \frac{n}{3} \rceil}$ maximale Cliquen.

Cliquen-Struktur

- Cliquen sind abgeschlossen unter Exklusion, d.h. wenn U eine Clique in G ist und v ein Knoten aus U, dann ist $U \setminus \{v\}$ auch eine Clique.
  Oder anders gesagt:
  Die Cliqueneigenschaft ist eine *hereditäre* Grapheigenschaft, denn sie vererbt sich auf induzierte Teilgraphen.

- Cliquen sind geschachtelt, d.h. jede Clique der Größe n enthält eine Clique der Größe $n-1$ (sogar n davon).
  (Das folgt hier sofort aus dem Abschluss unter Exklusion. Für andere Eigenschaften, die nicht unter Exklusion abgeschlossen sind, muss man das aber extra beweisen.)

Generalisierte Cliquen

Sei $G = (V, E)$ ein ungerichteter Graph, U eine Teilmenge der Knoten und $k > 0$ eine natürliche Zahl.
Generalisierte (distanz-basierte) Cliquen:

- U heißt k-Clique g.d.w. $\forall u, v \in U : \quad d_G(u, v) \leq k$.
- U heißt k-Club g.d.w. $\text{diam}(G[U]) \leq k$.
- U heißt k-Clan g.d.w. U ist eine maximale k-Clique und U ist ein k-Club.

- k-Cliquen sind nicht lokal definiert (die Distanzen können sich aus Pfaden ergeben, die über Knoten außerhalb von U führen).
- Obwohl k-Clubs und k-Clans lokal definiert sind (abgesehen von der Maximalitätsbedingung), sind sie nur von geringerem Interesse. Distanz-basierte Cliquen sind i.A. nicht abgeschlossen unter Exklusion und nicht geschachtelt.

Grundfunktionen

In $\mathcal{O}(m+n)$ können folgende Funktionen berechnet werden:

- Bestimme, ob eine gegebene Knotenteilmenge $U \subseteq V$ eine Clique in G ist.
  Bestimme für jede Kante in G, ob beide Endknoten in U sind.
  Zähle die Fälle und vergleiche mit $|U| \cdot (|U|-1)/2$.

- Bestimme, ob eine gegebene Clique $U \subseteq V$ maximal ist in G.
  Teste, ob es einen Knoten in $V \setminus U$ gibt, der adjazent zu allen Knoten in U ist.

Maximale Clique

Bestimme die lexikographisch kleinste maximale Clique U, die eine gegebene Clique $U' \subseteq V$ enthält.

- Annahme: V ist eine geordnete Menge
- Seien $U, U' \in V$. Def.: $U \leq U' \Leftrightarrow$ der kleinste Knoten der nicht in $U \cap U'$ ist, ist in U.
- Starte mit $U = U'$
- Itereriere über alle $v \in V \setminus U$ in aufsteigender Reihenfolge und teste, ob $U \subseteq N(v)$.
- Falls ja, dann füge v zu U hinzu.
- Am Ende ist U eine maximale Clique, die U' enthält.

$\Rightarrow$ ebenfalls $\mathcal{O}(m + n)$

Cliquen maximaler Kardinalität

Maximum-Clique: Clique der größtmöglichen Kardinalität in einem gegebenen Graphen

Primitiver Algorithmus: erschöpfende Suche
Zähle alle Kandidatensets $U \subseteq V$ auf
und bestimme, ob U eine Clique ist.
Gib die größte gefundene Clique aus.
$\Rightarrow$ Laufzeit $\mathcal{O}(n^2 \cdot 2^n)$

Clique-Problem

Entscheidungproblem:

Problem

Problem: ***Clique***
Eingabe: *Graph G, Parameter $k \in \mathbb{N}$*
Frage: *Existiert eine Clique der Kardinalität $\geq k$ in G?*

Härte des Clique-Problems

Sei $\omega(G)$ die Größe der Maximum-Clique(n) in G.
Wenn wir einen Algorithmus hätten, der **Clique** in Zeit $T(n)$ entscheidet, dann könnten wir $\omega(G)$ in Zeit $\mathcal{O}(T(n) \cdot \log n)$ mit binärer Suche berechnen.
Andererseits ergibt sich aus jedem Algorithmus zur Berechnung von $\omega(G)$ in Zeit $T(n)$ ein Algorithmus, der **Clique** in $T(n)$ entscheidet.
Ein polynomieller Algorithmus für das eine Problem würde als einen polynomiellen Algorithmus für das jeweils andere Problem implizieren.

Aber:

Satz

Clique *ist $\mathcal{NP}$-complete.*

Beweis: Reduktion von **Satisfiability** (Erfüllbarkeit)

Versteckte Cliquen

Folgerung

Falls $\mathcal{P} \neq \mathcal{NP}$ gilt, gibt es keinen Polynomialzeit-Algorithmus, der eine Clique der Größe k in einem Graphen findet, der garantiert eine solche Clique der Größe k enthält.

Bemerkung: die Schwierigkeit, eine versteckte Clique zu finden, hängt nicht von der Größe der Clique ab. Auch Cliquen der Größe $(1 - \varepsilon) \cdot n$ können nicht in Polynomialzeit gefunden werden.

Maximum-Clique: besserer exponentieller Algorithmen

Satz

Eine Clique maximaler Kardinalität kann in Zeit $\mathcal{O}^*(1.3803^n)$ berechnet werden.

($\mathcal{O}^*$ ignoriert polynomielle Faktoren)

Maximum-Clique: besserer exponentieller Algorithmus

Beweis.

- Sei v ein Knoten mit minimalem Grad $\deg_G(v) = \delta(G)$
- Falls $\delta(G) \geq n - 3$, dann
  - fehlen in G nur einfache Pfade und Kreise im Vergleich zum vollständigen Graphen K_n

  $\Rightarrow$ Maximum Clique kann in $\mathcal{O}(m + n)$ berechnet werden:
  - In einem Pfad P kann man die Größe einer maximalen unabhängigen Menge (independent set) berechnen als $\lceil |V(P)|/2 \rceil$.
  - In einem Kreis C ist die Größe $\lfloor |V(C)|/2 \rfloor$.
  - Da die Pfade und Kreise paarweise disjunkt sind, kann man die einzelnen Werte einfach addieren.

Maximum-Clique: besserer exponentieller Algorithmus

Beweis.

- Ansonsten sei v ein Knoten mit Grad $\deg_G(v) \leq n-4$
- Jede Maximum-Clique ist entweder
  - $\{v\}$ vereinigt mit einer Maximum-Clique von $G[N(v)]$ oder
  - eine Maximum-Clique von $G[V \setminus \{v\}]$.

$\Rightarrow$ Rekursive Berechnung mit worst-case-Zeit

$$T(n) \leq T(n-4) + T(n-1) + c \cdot (m+n)$$

$\Rightarrow$ $T(n) \in \mathcal{O}^*(\beta^n)$ mit $\beta \approx 1.3803$, wobei β die größte reelle Nullstelle des charakteristischen Polynoms $\beta^4 - \beta^3 - 1$ ist

Approximation von Maximum-Cliquen

Approximation der größten Clique mit Faktor $n/2$ ist einfach (wähle die Endknoten einer Kante, falls es eine gibt).

Satz

Es gibt einen Algorithmus, der bei Eingabe eines Graphen G mit n Knoten eine Clique ausgibt, deren Größe maximal um einen Faktor $\mathcal{O}\left(\frac{n}{(\log n)^2}\right)$ von der Maximum-Cliquengröße $\omega(G)$ abweicht.

Satz

Falls nicht $\mathcal{NP} = \mathcal{ZPP}$ gilt, dann existiert kein Polynomialzeitalgorithmus, der bei Eingabe eines Graphen G mit n Knoten eine Clique ausgibt, deren Größe maximal um einen Faktor $n^{1-\epsilon}$ von der Maximum-Cliquengröße $\omega(G)$ abweicht (für jedes $\epsilon > 0$).

Suche nach Cliquen fester Größe

- In einigen Fällen reicht es, nach Cliquen fester Größe zu suchen

$\Rightarrow$ Cliquengröße wird nicht als Teil der Eingabe betrachtet

- Vollständige Suche: $\mathcal{O}(k^2 \cdot n^k) = \mathcal{O}(n^k)$,
  wenn Cliquengröße k fest ist

Bessere Komplexität für Dreiecke

- $A(G)$: Adjazenzmatrix von Graph G
- $B(G) = A(G)^2 = A(G) \cdot A(G)$: Einträge b_{ij} sind die Wege der Länge 2 zwischen den Knoten v_i und v_j

$\Rightarrow$ läßt sich durch schnellere Matrixmultiplikation ausrechnen, z.B. in $\mathcal{O}(n^{2.376})$ (Coppersmith / Winograd, 1990)

- Existiert ein Eintrag $b_{ij} \geq 1$ mit $i \neq j$, dann gibt es einen Knoten $u \in V$, der zu v_i und zu v_j adjazent ist.
- Wenn nun auch eine Kante $\{v_i, v_j\}$ existiert, dann enthält der Graph ein Dreieck $\{v_i, v_j, u\}$

$\Rightarrow$ Checke für alle echt positiven Werte b_{ij}, ob es eine Kante $\{v_i, v_j\}$ gibt

$\Rightarrow$ Zeit-Komplexität: $\mathcal{O}(n^\alpha)$ mit $\alpha < 2.376$

Cliquen fester Größe $k \geq 3$

Sei $\alpha(r, s, t)$ so definiert, dass man die Multiplikation einer $n^r \times n^s$-Matrix mit einer $n^s \times n^t$-Matrix in Zeit $\mathcal{O}\left(n^{\alpha(r,s,t)}\right)$ berechnen kann.

Satz

Für jedes $k \geq 3$ existiert ein Algorithmus, der eine Clique der Größe k in einem Graphen mit n Knoten finden kann (falls eine solche existiert) und der in Zeit $\mathcal{O}\left(n^{\beta(k)}\right)$ läuft, wobei $\beta(k) = \alpha(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)$.

Cliquen fester Größe $k \geq 3$

Beweis.

- Seien $k_1 = \lfloor k/3 \rfloor$, $k_2 = \lceil (k-1)/3 \rceil$ und $k_3 = \lceil k/3 \rceil$
- Es gilt $k = k_1 + k_2 + k_3$.
- Konstruiere tripartiten Hilfsgraph $\tilde{G}$
  - $\tilde{V} = \tilde{V}_1 \cup \tilde{V}_2 \cup \tilde{V}_3$, wobei V_i aus allen Cliquen der Größe k_i in G besteht
  - Knoten $U \in \tilde{V}_i$ und $U' \in \tilde{V}_j$ sind adjazent in $\tilde{G}$ g.d.w. $i \neq j$ und $U \cup U'$ eine Clique der Größe $k_i + k_j$ in G ist.
- Teste $\tilde{G}$ auf Dreiecke

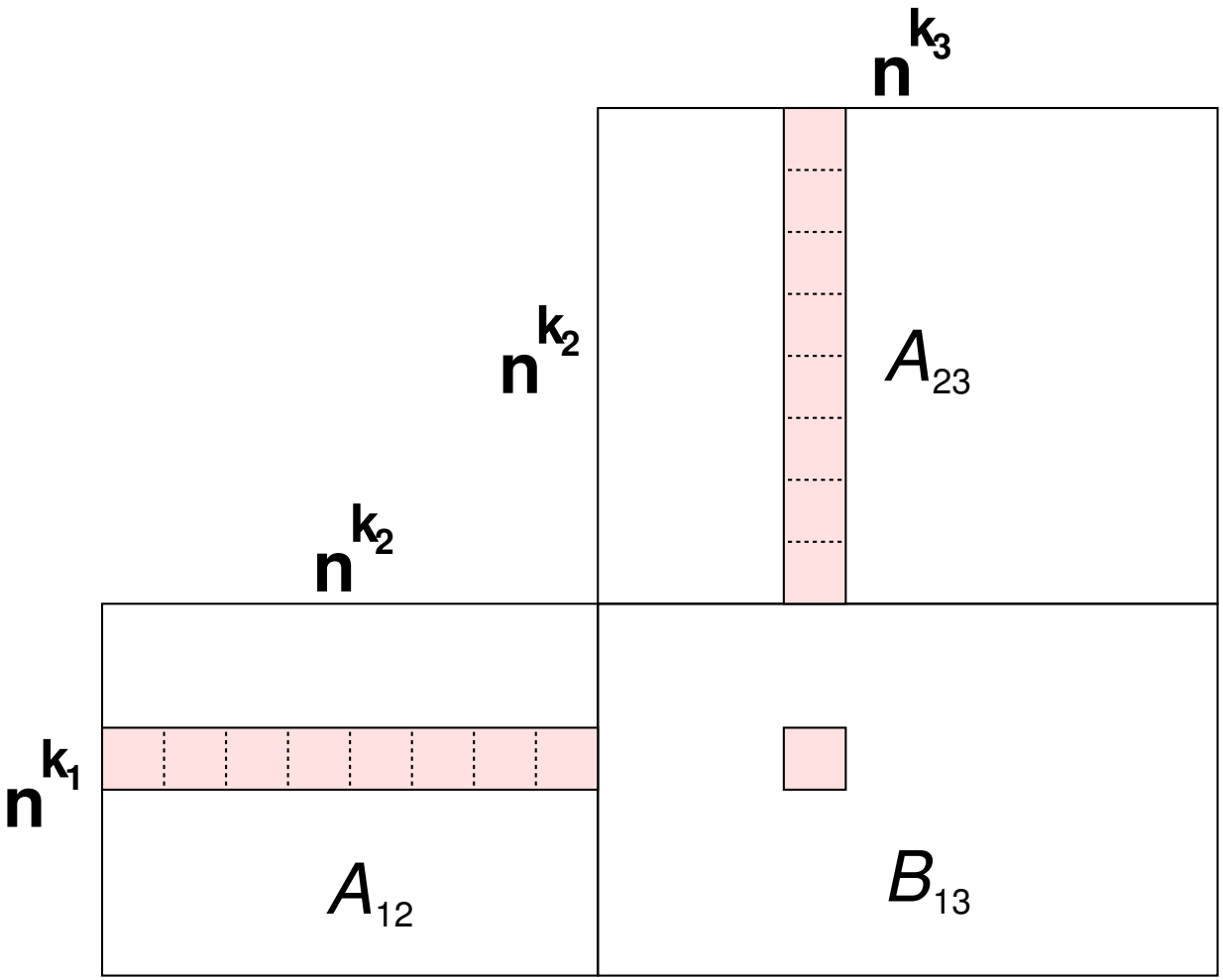
Cliquen fester Größe $k \geq 3$

Beweis.

- Dreieck $\{U_1, U_2, U_3\}$ impliziert eine Clique der Größe k in G
- geht mit schneller Matrixmultiplikation, aber hier muss eine $n^{k_1} \times n^{k_2}$-Matrix (Kanten zwischen $\tilde{V}_1$ und $\tilde{V}_2$) mit einer $n^{k_2} \times n^{k_3}$-Matrix (Kanten zwischen $\tilde{V}_2$ und $\tilde{V}_3$) multipliziert werden, und zwar in Zeit $\mathcal{O}\left(n^{\beta(k)}\right)$
- Berechnung der drei Matrizen A_{12}, A_{23} und A_{13} in $\mathcal{O}\left(n^{\max\{k_1+k_2, k_1+k_3, k_2+k_3\}}\right) = \mathcal{O}\left(n^{\lceil \frac{2k}{3} \rceil}\right)$

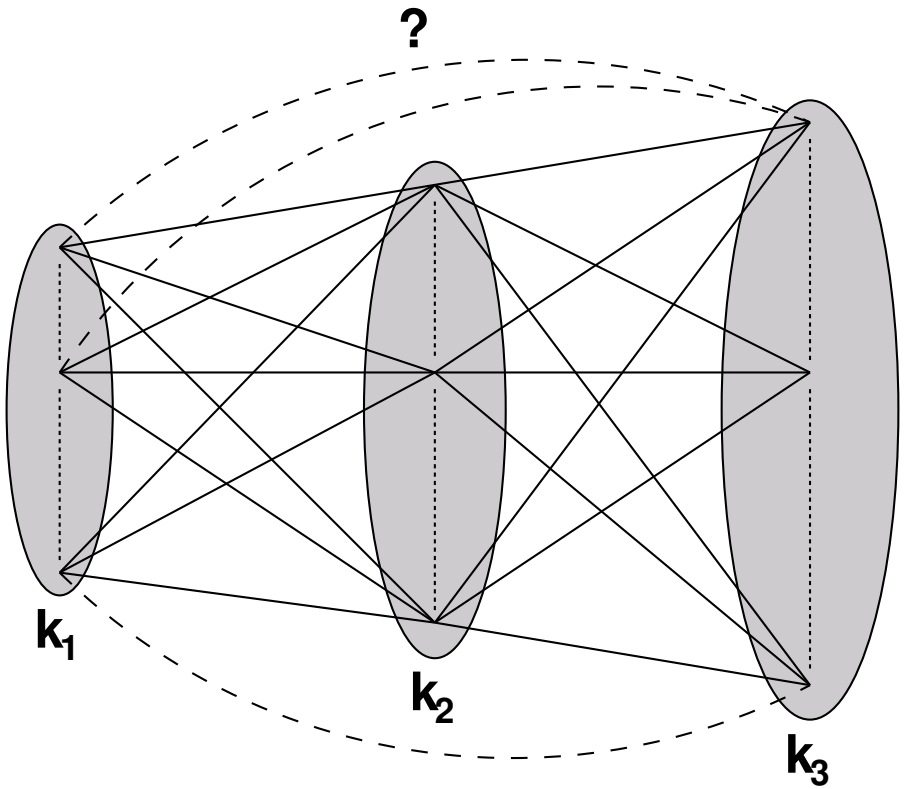
  (wird dominiert durch die Zeit $\mathcal{O}(n^{\beta(k)})$ für die Multiplikation der rechteckigen Matrizen, also $B_{13} = A_{12} \cdot A_{23}$)

Cliquen fester Größe $k \geq 3$



B_{13} wird dann mit A_{13} verknüpft

Cliquen fester Größe $k \geq 3$



Cliquen fester Größe: Beispielkomplexitäten

Cliquengröße	Vollständige Suche	Matrixmultiplikation
3	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{2.376})$
4	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{3.376})$
5	$\mathcal{O}(n^5)$	$\mathcal{O}(n^{4.220})$
6	$\mathcal{O}(n^6)$	$\mathcal{O}(n^{4.751})$
7	$\mathcal{O}(n^7)$	$\mathcal{O}(n^{5.751})$
8	$\mathcal{O}(n^8)$	$\mathcal{O}(n^{6.595})$

Cliquen fester Größe: Mitgliedszahlen

Satz

Für jedes $k \geq 3$ existiert ein Algorithmus, der in Zeit $\mathcal{O}\left(n^{\beta(k)}\right)$ läuft und der für jeden Knoten zählt, an wieviel Cliquen der Größe k er beteiligt ist (in einem Graphen mit n Knoten), wobei
$\beta(k) = \alpha(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)$.

Cliquen fester Größe: Mitgliedszahlen

Beweis.

- Für $k = 3$ (Dreiecke) kann man nicht nur feststellen, *ob* zwei Knoten v_i und v_j zu einem Dreieck gehören, sondern auch *zu wievielen*.
- Wenn Kante $\{v_i, v_j\}$ in G existiert, dann ist diese Anzahl gleich dem Eintrag b_{ij} in der quadrierten Adjazenzmatrix $B(G) = A(G) \cdot A(G)$.
- Anwendung im allgemeinen Fall von $\tilde{G}$:
  für jeden Knoten $v \in V$ sei $C_k(v)$ die Anzahl verschiedener Cliquen der Größe k, in denen v enthalten ist.
- Entsprechend sei $\tilde{C}_3(U)$ die Anzahl der Dreiecke in $\tilde{G}$, zu denen Knoten U gehört.
  (U ist eine Clique der Größe kleiner als k)

Cliquen fester Größe: Mitgliedszahlen

Beweis.

- Cliquen der Größe k können viele verschiedene Repräsentationen in $\tilde{G}$ haben.
- Diese Anzahl ist die Anzahl der Partitionierungen von einer Menge der Kardinalität k in drei Mengen der Kardinalitäten k_1, k_2 und k_3, also der Multinomialkoeffizient $\binom{k}{k_1,k_2,k_3}$.
- o.B.d.A. sei k_1 der kleinste der drei Parameter.
  Sei $\mathcal{U}(v)$ die Menge aller Cliquen U der Größe k_1 in G, so dass $v \in U$.
  Dann gilt:

$$\sum_{U \in \mathcal{U}(v)} \tilde{C}_3(U) = \binom{(k-1)}{(k_1-1), k_2, k_3} \cdot C_k(v)$$

- Berechne linke Seite in $\mathcal{O}\left(n^{\beta(k)}\right)$ (berechne Matrizen; suche Einträge für alle U, die v enthalten); Berechne $C_k(v)$)

'Effiziente' Aufzählungsalgorithmen

- Ausgabe oft exponentiell in der Eingabelänge

$\Rightarrow$ etwas anderer Effizienzbegriff

- polynomielle Gesamtzeit:
  bei Ausgabe von C Konfigurationen Begrenzung der Zeit durch ein Polynom in C und in der Eingabegröße n
  (sozusagen output-sensitiv)
  - vollständige Suche ist nicht in polynomieller Gesamtzeit
  - Aufzählung aller maximalen Cliquen geht in pol. Gesamtzeit
  - Aufzählung aller Maximum-Cliquen geht nur in pol. Ges.zeit, falls $\mathcal{P} = \mathcal{NP}$
- polynomielle Verzögerung (polynomial delay):
  Zeit bis zur Ausgabe der ersten Konfiguration, zwischen zwei aufeinanderfolgnden Ausgaben von Konfigurationen und von der Ausgabe der letzten Konfiguration bis zum Stop ist polynomiell in der Eingabegröße

Aufzählung maximaler Cliquen

Satz

Es gibt einen Algorithmus, der alle maximalen Cliquen mit (polynomieller) Verzögerung $\mathcal{O}\left(n^3\right)$ und (linearem) Platzverbrauch $\mathcal{O}(m+n)$ aufzählt.

Algorithmus zur Aufzählung maximaler Cliquen

- Konstruiere einen Binärbaum mit n Leveln, dessen Blätter sich nur auf Level n befinden.
- Jedes Level ist einem Knoten des Eingabegraphen G zugeordnet. Im Level i betrachtet man Knoten v_i.
- Insbesondere entsprechen die Knoten von Level i des Baums den maximalen Cliquen des induzierten Teilgraphen $G[\{v_1, \ldots, v_i\}]$.

$\Rightarrow$ Die Blätter sind genau die maximalen Cliquen von G.

- Für ein gegebenes Level i und eine maximale Clique U in $G[\{v_1, \ldots, v_i\}]$ wollen wir die Kinder auf dem nächsten Level $i+1$ bestimmen:
  1. Alle Knoten von U sind adjazent zu v_{i+1} in G.
  2. Es existiert ein Knoten in U, der nicht zu v_{i+1} adjazent ist in G

Algorithmus zur Aufzählung maximaler Cliquen

Fallunterscheidung:

1. Alle Knoten von U sind adjazent zu v_{i+1} in G.
   $\Rightarrow$ $U \cup \{v_{i+1}\}$ ist maximale Clique in $G[\{v_1, \ldots, v_{i+1}\}]$
   - Das ist die einzige Möglichkeit, eine maximale Clique in $G[\{v_1, \ldots, v_{i+1}\}]$ zu bekommen, die U enthält
     In diesem Fall hat U nur ein einziges Kind im Baum.
2. Es existiert ein Knoten in U, der nicht zu v_{i+1} adjazent ist in G
   Man kann 2 maximale Cliquen in $G[\{v_1, \ldots, v_{i+1}\}]$ erhalten:
   1. U ist selbst eine maximale Clique
   2. $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ ist eine Clique,
      wobei $\bar{N}(v_{i+1})$ alle nicht mit v_{i+1} adjazenten Knoten sind
      - Wenn die Menge eine *maximale* Clique ist, hätte U zwei Kinder im Baum
      - $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ könnte aber Kind von mehreren sein

      $\Rightarrow$ Kind der lexikographisch kleinsten Menge U (falls maximal)

$\Rightarrow$ Binärbaum
(interne Knoten haben 1 oder 2 Kinder, Blätter nur in Level n)

Algorithmus zur Aufzählung maximaler Cliquen

- Traversiere den Binärbaum per Tiefensuche (DFS)
- Ausgabe aller Blätter
- Gegeben Knoten U auf Level i:
  - `Parent`(U,i):
    Vaterknoten von U ist die lexikographisch kleinste maximale Clique in $G[\{v_1,\ldots,v_{i-1}\}]$, die $U \setminus \{v_i\}$ enthält
    $\Rightarrow$ Grundfunktion, in $\mathcal{O}(m+n)$
  - `LeftChild`(U,i):
    - falls $U \subseteq N(v_{i+1})$ (1. Fall), dann $U \cup \{v_{i+1}\}$
    - falls $U \not\subseteq N(v_{i+1})$ (Teil des 2. Falls), dann U
    - Unterscheidung der Fälle kostet $\mathcal{O}(m+n)$ Zeit
  - `RightChild`(U,i):
    - falls $U \subseteq N(v_{i+1})$, dann existiert kein rechtes Kind
    - falls $U \not\subseteq N(v_{i+1})$, dann
      $\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}$ falls es maximale Clique ist und
      $U = \texttt{Parent}\left(\left(U \setminus \bar{N}(v_{i+1})\right) \cup \{v_{i+1}\}, i+1\right)$
      sonst keins
    - Kosten: $\mathcal{O}(m+n)$ Zeit

Algorithmus zur Aufzählung maximaler Cliquen

- Längster Pfad zwischen zwei Blättern im Baum ist $2n-2$ und geht durch $2n-1$ Knoten
- pro Knoten Zeitaufwand $\mathcal{O}(m+n)$
- Jeder Unterbaum hat ein Blatt auf Level n

$\Rightarrow$ Ausgabeverzögerung $\mathcal{O}(n^3)$

- Wenn ein Knoten bearbeitet wird, muss nur die Menge U, das Level i, sowie ein Label zur Unterscheidung von linkem/rechten Kind gespeichert werden

$\Rightarrow$ Speicheraufwand $\mathcal{O}(m+n)$

Aufzählung in lexikographischer Reihenfolge

- Es ist $\mathcal{NP}$-vollständig für einen Graphen G und eine maximale Clique U von G zu entscheiden, ob es eine maximale Clique U' gibt, die lexikographisch größer ist als U.
- Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen Algorithmus, der in Polynomialzeit zu einem Graphen G und einer maximalen Clique U von G die lexikographisch nächstgrößere maximale Clique generiert.
- Überraschenderweise kann man trotzdem alle maximalen Cliquen in lexikographischer Ordnung mit polynomieller Verzögerung aufzählen.

Aufzählung in lexikographischer Reihenfolge

- Idee: während der Generierung der aktuellen Ausgabe wird zusätzliche Arbeit in die Erzeugung lexikographisch größerer Cliquen investiert

$\Rightarrow$ Diese werden in einer Priority Queue gespeichert, die dann u.U. exponentiell viele Cliquen enthält und damit auch exponentiell viel Speicherplatz braucht.

Aufzählung in lexikographischer Reihenfolge

Algorithmus 3 : Alg. von Johnson, Papadimitriou und Yannakakis

$U_0 :=$ erste (lexikographisch kleinste) maximale Clique;
Füge U_0 in Priority Queue Q ein;
while Q *ist nicht leer* **do**
    $U :=$ExtractMin(Q);
    Ausgabe U;
    foreach *Knoten* v_j *von* G, *der zu einem Knoten* $v_i \in U$ *mit* $i < j$ *nicht adjazent ist* **do**
        $U_j := U \cap \{v_1, \ldots, v_j\}$;
        if $(U_j - \overline{N}(v_j)) \cup \{v_j\}$ *ist eine maximale Clique in* $G[\{v_1, \ldots, v_j\}]$ **then**
            Sei T die lexikographisch kleinste maximale Clique, die $(U_j - \overline{N}(v_j)) \cup \{v_j\}$ enthält;
            Füge T in Q ein

Aufzählung in lexikographischer Reihenfolge

Satz

Der Algorithmus von Johnson, Papadimitriou und Yannakakis zählt alle maximalen Cliquen eines Graphen mit n Knoten in lexikographischer Reihenfolge und mit Delay $\mathcal{O}(n^3)$ auf.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Korrektheit:

- Menge T ist beim Einfügen in Q (bei Betrachtung von U) lexikographisch größer als U
  (denn es wird ja aus U zumindest Knoten v_i entnommen während lediglich v_j hinzukommt und es gilt $i < j$)

$\Rightarrow$ Es werden nur Mengen in der Queue gespeichert, die erst nach U ausgegeben werden dürfen.

$\Rightarrow$ Die ausgegebenen maximalen Cliquen sind lexikographisch aufsteigend.

Aufzählung in lexikographischer Reihenfolge

Beweis.

Vollständigkeit:

- Falls U die lexikographisch kleinste noch auszugebende Clique ist, dann ist U in Q.
- Induktionsanfang: Für $U = U_0$ ist das korrekt.
- Induktionsschritt: Sei U lexikographisch größer als U_0.
  - Sei j der größte Index, dass $U_j = U \cap \{v_1, \ldots, v_j\}$ *keine* maximale Clique in $G[\{v_1, \ldots, v_j\}]$.
    (Muss existieren, weil sonst $U = U_0$. Außerdem muss $j < n$ sein, weil U eine maximale Clique des Gesamtgraphen G ist.)
  - Aufgrund der Maximalität von j muss gelten $v_{j+1} \in U$.
  - $\exists$ Menge S: $U_j \cup S$ ist maximale Clique in $G[\{v_1, \ldots, v_j\}]$

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):
  - Aufgrund der Maximalität von j ist v_{j+1} nicht adjazent zu allen Knoten in S.

  $\Rightarrow$ $\exists$ maximale Clique U', die zwar $U_j \cup S$, aber nicht v_{j+1} enthält
  - $U' \leq U$, weil sie sich in S unterscheiden
  - U' wurde schon ausgegeben (Ind.voraussetzung)
  - Als U' ausgegeben wurde, wurde festgestellt, dass v_{j+1} nicht adjazent ist zu einem Knoten $v_i \in U'$ mit $i < j+1$
  - $(U'_{j+1} \setminus \bar{N}(v_{j+1})) \cup \{v_{j+1}\} = U_{j+1}$
    und U_{j+1} ist maximale Clique in $G[\{v_1, \ldots, v_{j+1}\}]$

  $\Rightarrow$ Die lexikographisch kleinste maximale Clique, die U_{j+1} enthält, wurde in Q eingefügt.

Aufzählung in lexikographischer Reihenfolge

Beweis.

- Induktionsschritt (Fortsetzung):
  - Aufgrund der Maximalität von j stimmen U und T in den ersten $j+1$ Knoten überein.
  - Annahme: $U \neq T$
    Sei k der kleinste Index eines Knoten v_k, der in genau einer der beiden Mengen ist.
  - $k > j+1$
  - Da $T \leq U$, gilt $v_k \in T$ und $v_k \notin U$.

  $\Rightarrow$ U_k ist nicht maximale Clique in $G[\{v_1, \ldots, v_k\}]$
  (Widerspruch zur Maximalität von j)

  $\Rightarrow$ $U = T$

  $\Rightarrow$ U ist in der Priority Queue Q

Aufzählung in lexikographischer Reihenfolge

Komplexität:

- Extraktion der lexikographisch kleinsten maximalen Clique aus Q: $\mathcal{O}(n \log C)$
- n Berechnungen von maximalen Cliquen, die eine gegebene Menge enthalten: $\mathcal{O}(m+n)$ pro Menge
- Einfügen einer maximalen Clique in Q: $\mathcal{O}(n \log C)$ pro Clique
- Da $C \leq 3^{\lceil \frac{n}{3} \rceil}$, folgt dass die Verzögerung $\mathcal{O}(n^3)$ ist.

Plex

Relaxiere Cliquenbegriff, so dass konstant viele Verbindungen zu Gruppenmitgliedern bei jedem Knoten fehlen dürfen.

Definition

Sei $G = (V, E)$ ein ungerichteter Graph und sei $k \in \{1, \ldots, n-1\}$ eine natürliche Zahl.
Dann wird ein Subset $U \subseteq V$ als k-Plex bezeichnet, falls $\delta(G[U]) \geq |U| - k$.

k-Plex-Eigenschaften

- Jede Clique ist ein 1-Plex.
- Jedes k-Plex ist auch ein $(k+1)$-Plex.
- Ein *maximales k-Plex* ist in keinem größeren k-Plex echt enthalten.
- Ein *Maximum k-Plex* hat maximale Kardinalität unter allen k-Plexen in G.
- Jeder induzierte Teilgraph eines k-Plex ist auch ein k-Plex, d.h. die k-Plex-Eigenschaft ist abgeschlossen unter Exklusion.

k-Plex-Durchmesser

Satz

Sei $k \in \{1, \ldots, n-1\}$ eine natürliche Zahl und sei $G = (V, E)$ ein ungerichteter Graph auf n Knoten. Dann gilt:

1. *Wenn V ein k-Plex mit $k < \frac{n+2}{2}$ ist, dann gilt $\mathrm{diam}(G) \leq 2$. Falls zusätzlich $n \geq 4$, dann ist G zweifach kantenzusammenhängend.*
2. *Wenn V ein k-Plex mit $k \geq \frac{n+2}{2}$ und G zusammenhängend ist, dann gilt $\mathrm{diam}(G) \leq 2k - n + 2$.*

k-Plex-Durchmesser

Beweis.

Fall $k < \frac{n+2}{2}$:

- Seien $u, v \in V$ nicht adjazente Knoten $(u \neq v)$
  (adjazente Knoten haben $d(u, v) = 1$)
- Annahme: $d(u, v) \geq 3 \Rightarrow N(u) \cap N(v) = \emptyset$, d.h.

$$n - 2 \geq |N(u) \cup N(v)| \geq 2\delta(G) \geq 2(n - k) > \ldots$$

$$\ldots > 2\left(n - \frac{n+2}{2}\right) = n - 2$$

(Widerspruch)

$\Rightarrow$ $d(u, v) \leq 2 \quad \Rightarrow \quad \text{diam}(G) \leq 2$

k-Plex-Durchmesser

Beweis.

Fall $k < \frac{n+2}{2}$, $n \geq 4$:

- Annahme: $\exists$ Brücke e (d.h. $G - \{e\}$ enthält zwei Zusammenhangskomponenten V_1 und V_2)
- Jeder kürzeste Pfad von einem Knoten in V_1 zu einem Knoten in V_2 muss diese Brücke benutzen.

$\Rightarrow$ Da $\text{diam}(G) \leq 2$, muss eine Komponente ein einzelner Knoten sein. Dieser Knoten hat Grad 1.

- Da V ein k-Plex mit $n \geq 4$ Knoten ist, gilt für den Grad dieses Knotens v: $\deg(v) \geq n - k > n - \frac{n+2}{2} = \frac{n-2}{2} \geq 1$ (Widerspruch)

$\Rightarrow$ Es existiert keine Brücke in G bzw. G ist 2-fach kantenzusammenhängend.

k-Plex-Durchmesser

Beweis.

Fall $k \geq \frac{n+2}{2}$:

- Sei $\{v_0, v_1, \ldots, v_r\}$ ein längster kürzester Pfad, also ein Pfad mit $d(v_0, v_r) = r = \text{diam}(G)$.
- Wir können annehmen, dass $r \geq 4$.
- Da es keinen kürzeren Pfad zwischen v_0 und v_r gibt, ist v_i zu keinem der Knoten $v_0, \ldots, v_{i-2}, v_{i+2}, \ldots, v_r$ auf dem Pfad adjazent, außer zu seinen Pfad-Nachbarknoten v_{i-1} und v_{i+1}
- Außerdem kann kein Knoten existieren, der gleichzeitig zu v_0 und zu v_3 adjazent ist.

$\Rightarrow \{v_0\} \uplus \{v_2, v_3, \ldots, v_r\} \uplus (N(v_3) \setminus \{v_2, v_4\}) \subseteq \overline{N}(v_0)$

$\Rightarrow 1 + (r-1) + d_G(v_3) - 2 \leq k \quad \Rightarrow \quad r + n - k - 2 \leq k$

$\Rightarrow \text{diam}(G) = r \leq 2k - n + 2$

k-Plex Problem

- (variables) Entscheidungsproblem **Plex** mit Eingabe
  - Graph G,
  - Größenparameter ℓ und
  - Plex-Parameter k

  ist $\mathcal{NP}$-vollständig, da das **Clique**-Problem sich darauf reduzieren lässt (mit $k = 1$)

$\Rightarrow$ Betrachte das Problem für feste Plex-Parameter c

Problem

Problem: *c-**Plex***
Eingabe: *Graph G, Parameter $\ell \in \mathbb{N}$*
Frage: *Existiert ein c-Plex der Kardinalität $\geq \ell$ in G?*

c-Plex

Genau wie 1-**Plex**=**Clique** sind auch alle anderen Probleme für einen festen Plex-Parameter c $\mathcal{NP}$-vollständig:

Satz

*c-**Plex** ist $\mathcal{NP}$-vollständig für alle $c \in \mathbb{N}$.*

Beweis: durch Reduktion von **Clique**, siehe z.B. Brandes/Erlebach (Eds.): Network Analysis (S. 128).

Cores

Relaxiere Cliquenbegriff, so dass konstant viele Verbindungen zu Gruppenmitgliedern bei jedem Knoten mindestens vorhanden sein müssen.

Definition

Sei $G = (V, E)$ ein ungerichteter Graph und sei $k \in \{1, \ldots, n-1\}$ eine natürliche Zahl.
Dann wird ein Subset $U \subseteq V$ als k-Core (k-Kern) bezeichnet, falls $\delta(G[U]) \geq k$.
Ein *maximaler k-Core* ist in keinem größeren k-Core echt enthalten.
Ein *Maximum k-Core* hat maximale Kardinalität unter allen k-Cores in G.
Jeder Maximum-k-Core ist auch ein maximaler k-Core (was umgekehrt nicht unbedingt gilt).

k-Core-Eigenschaften

- Jeder Graph G ist ein $\delta(G)$-Core, jede Clique ist ein $n-1$-Core.
- Jeder $(k+1)$-Core ist auch ein k-Core.
- Jeder k-Core ist ein $(n-k)$-Plex.
- Wenn U und U' k-Cores sind, dann ist auch $(U \cup U')$ ein k-Core.

$\Rightarrow$ Maximale k-Cores sind einzigartig.

- Nicht jeder induzierte Teilgraph eines k-Cores ist auch wieder ein k-Core, d.h. die k-Core-Eigenschaft ist *nicht* abgeschlossen unter Exklusion.
  Bsp.: Jeder Kreis ist ein 2-Core, aber jeder echte Teilgraph enthält nmindestens einen Knoten vom Grad < 2.
- k-Cores sind auch nicht geschachtelt. (Nicht jeder k-Core der Größe n enthält einen k-Core der Größe $n-1$.)
- k-Cores müssen nicht unbedingt zusammenhängend sein.

Maximale zusammenhängende k-Cores

Fakt

Sei $G = (V, E)$ ein ungerichteter Graph und $k > 0$ eine natürliche Zahl. Wenn U und U' zwei maximale zusammenhängende k-Cores in G mit $U \neq U'$ sind, dann gibt es keine Kante zwischen U und U'.

Folgerung

- *Der einzige Maximum k-Core eines Graphen ist die Vereinigung seiner maximalen zusammenhängenden k-Cores.*
- *Der Maximum 2-Core eines zusammenhängenden Graphen ist zusammenhängend.*
  (Wenn er es nicht wäre, könnte man die Teile verbinden und alle neuen Knoten hätten mindestens Grad 2.)
- *Ein Graph ist genau dann ein Wald, wenn er keine 2-Cores enthält.*

Der Maximum k-Core

Satz

Sei $G = (V, E)$ ein ungerichteter Graph und $k > 0$ eine natürliche Zahl. Wenn man wiederholt alle Knoten mit Grad $< k$ (und alle inzidenten Kanten) entfernt, dann entspricht die verbleibende Knotenmenge U genau dem Maximum k-Core.

Beweis.

- U ist ein k-Core, d.h. wir müssen nur noch Maximum zeigen.
- Annahme: U ist nicht Maximum k-Core.

$\Rightarrow$ $\exists$ nichtleere Menge $T \subseteq V$, so dass $U \cup T$ Maximum k-Core

- Als der erste Knoten von T aus G entfernt wurde, muss er Grad $< k$ gehabt haben. Das kann aber nicht sein, da er mindestens k Nachbarn in $U \cup T$ hat und alle anderen Knoten noch im Graph enthalten waren. (Widerspruch)

□

Core-Zerlegung

Definition

Die Core-Zahl $\xi_G(v)$ eines Knotens $v \in V$ ist die höchste Zahl k, so dass v Teil eines k-Cores im Graphen G ist, d.h.

$$\xi_G(v) = \max\{k : \exists k\text{-Core } U \text{ in } G \text{ mit } v \in U\}$$

Algorithmus zur Berechnung arbeitet nach folgendem Prinzip:

- Jeder Graph G ist ein $\delta(G)$-Core.
- Jeder Nachbarknoten mit geringerem Grad verringert die potentielle Core-Zahl eines Knotens.

Berechnung der Core-Zahlen

Algorithmus 4 : Berechnung der Core-Zahlen

Input : Graph $G = (V, E)$

Output : Array ξ_G mit den Core-Zahlen aller Knoten in G

Berechne die Grade aller Knoten und speichere sie in D;
Sortiere V in aufsteigender Grad-Folge D;
foreach $v \in V$ in sortierter Reihenfolge **do**
    $\xi_G(v) := D[v]$;
    foreach *Knoten u adjazent zu v* **do**
        if $D[u] > D[v]$ **then**
            $D[u] := D[u] - 1$;
            Sortiere V in aufsteigender Grad-Reihenfolge nach D

Berechnung der Core-Zahlen

Komplexität:

- naiv: $\mathcal{O}(mn \log n)$
  (teuerste Operationen: Sortieren der Knoten nach dem Grad)
- besser: $\mathcal{O}(m + n)$
  - Knotengrade haben Werte aus dem Intervall $[0, n-1]$
    $\Rightarrow$ Sortieren mit n Buckets in $\mathcal{O}(n)$
  - Nachsortieren kann man sich sparen, indem man sich merkt an welchen Indizes im Array die Knoten des nächsten Grads beginnen
    Beim dekrementieren des Werts eines Knotens kommt der Knoten einfach an den Anfang des alten Intervalls und dann wird die Grenze um eine Stelle verschoben, so dass er nun zum Intervall des kleineren Grads gehört (das geht in $\mathcal{O}(1)$).
  - insgesamt: $\mathcal{O}(n)$ für Initialisierung / Sortieren,
    dann $\mathcal{O}(m)$ für die Schleifendurchläufe (jede Kante wird höchstens zweimal betrachtet), also $\mathcal{O}(m + n)$

Dichte Subgraphen

Definition

Sei $G = (V, E)$ ein ungerichteter Graph mit n Knoten und m Kanten. Die Dichte $\rho(G)$ des Graphen G ist definiert als

$$\rho(G) = \frac{m}{\binom{n}{2}}$$

Ein durch eine Knotenteilmenge $U \subseteq V$ eines Graphen $G = (V, E)$ induzierter Teilgraph heißt η-dicht (für eine reelle Zahl η mit $0 \leq \eta \leq 1$), falls gilt:

$$\rho(G[U]) \geq \eta$$

Aber: selbst (Teil-)Graphen mit hoher Dichte können isolierte Knoten enthalten!

Eigenschaften

- Cliquen sind 1-dichte (Teil-)Graphen.
- Ein k-Plex hat Dichte $1 - \frac{k-1}{n-1}$

$\Rightarrow$ Für $n \to \infty$ gilt für k-Plexe $\eta \to 1$) bzw.
$\forall k > 0, 0 \leq \eta \leq 1$: ein k-Plex der Größe mindestens $\frac{k-\eta}{1-\eta}$ ist ein η-dichter (Teil-)Graph.

- Aber: nicht jeder $1 - \frac{k-1}{n-1}$-dichte (Teil-)Graph ist auch ein k-Plex.
- Ein k-Core ist ein $\frac{k}{n-1}$-dichter (Teil-)Graph.
  Die Dichte von k-Cores kann sich für $n \to \infty$ beliebig nah an 0 annähern.
- η-dichte Graphen sind nicht abgeschlossen unter Exklusion (Knotenausschluss), sie sind aber geschachtelt.

Schachtelung von η-dichten Graphen

Satz

Sei η eine reelle Zahl mit $0 \leq \eta \leq 1$. Dann gilt: Ein η-dichter Teilgraph der Größe ℓ eines Graphen G enthält einen η-dichten Teilgraph der Größe $\ell - 1$ in G.

Schachtelung von η-dichten Graphen

Beweis.

- Sei
  - U ein η-dichter Teilgraph von G mit $|U| = \ell$,
  - m_U die Anzahl der Kanten in $G[U]$,
  - v ein Knoten mit minimalem Grad in $G[U]$ (also $\deg_{G[U]}(v) = \delta(G[U])$)
- $\delta(G[U]) \leq \overline{d}(G[U]) = \frac{2m_U}{\ell} = \rho(G[U])(\ell - 1)$
- Betrachte Knotenteilmenge $U' = U \setminus \{v\}$ mit Kantenanzahl $m_U - \delta(G[U]) \geq \rho(G[U])\binom{\ell}{2} - \rho(G[U])(\ell - 1) = \rho(G[U])\binom{\ell-1}{2}$

$\Rightarrow$ $\rho(G[U']) \geq \rho(G[U]) \geq \eta$

$\Rightarrow$ U' ist ein η-dichter Teilgraph der Größe $\ell - 1$.

Dichte und Wege

- Definition der Dichte entspricht der durchschnittlichen Existenz von möglichen Kanten innerhalb eines (induzierten) Teilgraphen
- Kanten sind Wege der Länge 1
- verallgemeinert kann man Wege (Walks) beliebiger Länge betrachten
- Grad der Ordnung $\ell \in \mathbb{N}$ eines Knotens v,
  Anzahl der Wege der Länge ℓ in G, die in v starten: $w_G^\ell(v)$
- $w_G^0(v) = 1$, $w_G^1(v) = \deg_G(v)$
- Anzahl Wege der Länge ℓ in einem Graphen G: w_G^ℓ

Anzahl Wege der Länge ℓ

Satz

Sei $G = (V, E)$ ein ungerichteter Graph.
Für alle $\ell \in \mathbb{N}$ und für alle $r \in \{0, \ldots, \ell\}$ gilt:

$$w_\ell(G) = \sum_{v \in V} w_G^r(v) \cdot w_G^{\ell - r}(v)$$

Anzahl Wege der Länge ℓ

Beweis.

- Jeder Weg der Länge ℓ besteht aus (nicht unbedingt verschiedenen) Knoten $v_0, \ldots, v_\ell$. Betrachte nun von jedem solchen Weg den Knoten v_r (für ein festgelegtes $r \in \{0, \ldots, \ell\}$).
- Es gibt $w_G^r(v_r)$ verschiedene Pfade der Länge r, die in v_r enden, und es gibt $w_G^{\ell-r}(v_r)$ verschiedene Pfade, die in v_r beginnen.

$\Rightarrow$ Es gibt also $w_G^r(v_r) \cdot w_G^{\ell-r}(v_r)$ verschiedene Pfade der Länge ℓ, die an der Stelle r den Knoten v_r besuchen.

$\Rightarrow$ Die Summe ergibt genau die Anzahl aller Pfade der Länge ℓ, da die entsprechenden Wege der einzelnen Summanden sich im Knoten an der Stelle r unterscheiden und somit nichts doppelt gezählt wird.

□

Dichte der Ordnung ℓ

- Maximale Anzahl der Wege der Länge ℓ in einem Graphen mit n Knoten: $n \cdot (n-1)^\ell$
- Dichte der Ordnung ℓ:

$$\rho_\ell(G) = \frac{w_\ell(G)}{n \cdot (n-1)^\ell}$$

- $\rho_1(G) = \rho(G)$, denn in $w_1(G)$ wird jede Kante doppelt gezählt.

Monotonität der Dichte

Satz

Für alle Graphen G und alle natürlichen Zahlen $\ell \geq 2$ gilt:

$$\rho_\ell(G) \leq \rho_{\ell-1}(G)$$

Beweis.

Da gilt $w_\ell(G) = \sum_{v \in V} w_G^r(v) \cdot w_G^{\ell-r}(v)$, gilt insbesondere für $r = 1$:

$$\begin{aligned} w_\ell(G) &= \sum_{v \in V} w_G^1(v) \cdot w_G^{\ell-1}(v) \\ &\leq (n-1) \cdot \sum_{v \in V} w_G^{\ell-1}(v) = (n-1) \cdot w_{\ell-1}(G) \end{aligned}$$

η-dichte Teilgraphen der Ordnung ℓ

Definition

In einem Graphen $G = (V, E)$ bezeichnet man den durch eine Knotenteilmenge $U \subseteq V$ induzierten Teilgraphen genau dann als η-dichten Teilgraphen der Ordnung ℓ, wenn gilt

$$\rho_\ell(G[U]) \geq \eta$$

- Aus dem Monotonitätssatz folgt, dass jeder η-dichte Teilgraph der Ordnung ℓ auch ein η-dichter Teilgraph der Ordnung $\ell - 1$ ist.
- Die η-dichten Teilgraphen der Ordnung $\ell \geq 2$ sind (analog zu den η-dichten Teilgraphen) geschachtelt.
- Wenn man eine Dichte η festlegt und die η-dichten Graphen wachsender Ordnung betrachtet, dann werden diese der Clique immer ähnlicher.

Dichte unendlicher Ordnung

Definition

Die *Dichte unendlicher Ordnung* ist

$$\rho_\infty(G) = \lim_{\ell \to \infty} \rho_\ell(G)$$

Satz

Sei $G = (V, E)$ ein ungerichteter Graph

1. *$\rho_\infty(G)$ ist entweder Null oder Eins.*
2. *G ist genau dann eine Clique, wenn $\rho_\infty = 1$.*

- Die einzigen Graphen, die für ein $\eta > 0$ und für jede Ordnung ℓ η-dicht der Ordnung ℓ sind, sind die Cliquen.
- Die Ordnung erlaubt eine gewisse Skalierung, wie wichtig Kompaktheit im Vergleich zu Dichte ist.

Durchschnittsgrad

- Dichte und Durchschnittsgrad eines Graphen hängen unmittelbar zusammen:
$$\bar{d}(G) = \rho(G) \cdot (n-1)$$
- Ein k-dichter (Teil-)Graph (bzgl. Durchschnittsgrad) kann definiert werden als (Teil-)Graph mit Durchschnittsgrad $\geq k$.
- η-dichte Graphen (bzgl. prozentualer Dichte) der Größe ℓ sind $\eta(\ell-1)$-dichte Graphen (bzgl. Durchschnittsgrad).
- k-dichte Graphen (bzgl. Durchschnittsgrad) der Größe ℓ sind $\frac{k}{\ell-1}$-dichte Graphen (bzgl. prozentualer Dichte).
- Jeder k-Core ist ein k-dichter (Teil-)Graph.
- k-dichte Teilgraphen sind nicht abgeschlossen unter Exklusion und auch nicht geschachtelt.
(Löscht man aus einem k-regulären Graph einen Knoten, fällt der Durchschnittsgrad unter k.)

Verallgemeinerung des Satzes von Turán

Satz (Dirac, 1963)

Sei $G = (V, E)$ ein ungerichteter Graph.
Wenn $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, dann enthält G einen Teilgraph der Größe $k + r$ mit Durchschnittsgrad $\geq k + r - 1 - \frac{r}{k+r}$ für alle $r \in \{0, \ldots, k-2\}$ und $n \geq k + r$.

(Der Fall $r = 0$ entspricht dem Satz von Turán.)

Der größtmögliche Durchschnittsgrad

$$\gamma^*(G) = \max_{U \subseteq V, U \neq \emptyset} \{\overline{\deg}(G[U])\}$$

Problem

Problem: ***DensestSubgraph***
Eingabe: *Graph G*
Frage: *Eine Knotenmenge mit maximalem induzierten Durchschnittsgrad*

Satz

Das Problem ***DensestSubgraph*** *kann für Graphen mit n Knoten und m Kanten in Zeit $\mathcal{O}\left(mn(\log n)\left(\log \frac{n^2}{m}\right)\right)$ gelöst werden.*

Der größtmögliche Durchschnittsgrad

- Formulierung von **DensestSubgraph** als MaximumFlow-Problem mit Parameter $\gamma \in \mathbb{Q}^+$
- Sei $G = (V, E)$ ein ungerichteter Graph mit n Knoten und m Kanten.
- Betrachte Flussnetzwerk bestehend aus Graph $G' = (V', E')$ und Kapazitätsfunktion $u_\gamma : E' \to \mathbb{Q}^+$
- Füge zu V eine Quelle s und eine Senke t hinzu:

$$V' = V \cup \{s, t\}$$

Der größtmögliche Durchschnittsgrad

- Konstruiere die neuen Kanten wie folgt:
  - ersetze jede (ungerichtete) Kante von G durch zwei gerichtete Kanten der Kapazität 1,
  - verbinde die Quelle mit allen Knoten in V durch eine Kante der Kapazität m,
  - verbinde alle Knoten in V mit der Senke durch eine Kante der Kapazität $m + \gamma - \deg_G(v)$.

$$\begin{aligned} E' &= \{(v,w) : \{v,w\} \in E\} \cup \\ &\quad \{(s,v) : v \in V\} \cup \\ &\quad \{(v,t) : v \in V\}\} \end{aligned}$$

$$u_\gamma(v,w) = \begin{cases} 1 & \text{falls } \{v,w\} \in E \\ m & \text{falls } v = s \\ m + \gamma - \deg_G(v) & \text{falls } w = t \\ 0 & \text{falls } (v,w) \notin E' \end{cases}$$

Der größtmögliche Durchschnittsgrad

- Wir betrachten Schnitt-Kapazitäten im Netzwerk.
- Sei S, T eine Partitionierung der Knotenmenge V' in zwei disjunkte Menge mit $s \in S$ und $t \in T$.
- Definiere $S_+ = S \setminus \{s\}$ und $T_+ = T \setminus \{t\}$.

$\Rightarrow$ $S_+ \cup T_+ = V$

Der größtmögliche Durchschnittsgrad

Falls $S_+ = \emptyset$, dann ist die Kapazität des Schnitts $c(S, \bar{S}) = m|V| = mn$, ansonsten erhält man

$$
\begin{aligned}
c(S,T) &= \sum_{v \in S, w \in T} u_\gamma(v,w) \\
&= \sum_{w \in T_+} u_\gamma(s,w) + \sum_{v \in S_+} u_\gamma(v,t) + \sum_{v \in S_+, w \in T_+} u_\gamma(v,w) \\
&= m|T_+| + \left(m|S_+| + \gamma|S_+| - \sum_{v \in S_+} \deg_G(v) \right) + \sum_{\substack{v \in S_+, w \in T_+ \\ \{v,w\} \in E}} 1 \\
&= m|V| + |S_+| \left(\gamma - \frac{1}{|S_+|} \left(\sum_{v \in S_+} \deg_G(v) - \sum_{\substack{v \in S_+, w \in T_+ \\ \{v,w\} \in E}} 1 \right) \right) \\
&= m|V| + |S_+|(\gamma - \overline{\deg}(G[S_+]))
\end{aligned}
$$

Der größtmögliche Durchschnittsgrad

- γ ist der Test-Wert für den maximalen Durchschnittsgrad.
- Wie kann man nun feststellen, ob er zu klein oder zu groß ist?

Satz

Seien S und T so gewählt, dass sie einem Minimum-s,t-Schnitt für γ entsprechen. Dann gilt:

1. *Wenn $S_+ \neq \emptyset$, dann $\gamma \leq \gamma^*(G)$.*
2. *Wenn $S_+ = \emptyset$, dann $\gamma \geq \gamma^*(G)$.*

Der größtmögliche Durchschnittsgrad

Beweis.

1. $S_+ \neq \emptyset$:
   Da $c(\{s\}, V' \setminus \{s\}) = m|V| \geq c(S,T)$, gilt
   $|S_+|(\gamma - \overline{\deg}(G[S_+])) \leq 0$.
   Also $\gamma \leq \overline{\deg}(G[S_+]) \leq \gamma^*(G)$.
2. $S_+ = \emptyset$: Annahme: $\gamma < \gamma^*(G)$
   Sei $U \subseteq V$ eine nichtleere Menge mit $\overline{\deg}(G[U]) = \gamma^*(G)$.
   Mit der Gleichung für $c(S,T)$ erhält man

$$c(U \cup \{s\}, \overline{U} \cup \{t\}) = mn + |U|(\gamma - \gamma^*(G)) < mn = c(S,T)$$

   (Widerspruch zur Minimalität der Schnittkapazität $c(S,T)$)
   $\Rightarrow \gamma \geq \gamma^*(G)$

Der größtmögliche Durchschnittsgrad

- Algorithmus benutzt binäre Suche, um den richtigen Wert für γ zu finden
- $\gamma^*(G)$ kann nur endlich viele Werte annehmen:

$$\gamma^*(G) \in \left\{ \frac{2i}{j} \ : \ i \in \{0, \ldots, m\} \text{ und } j \in \{1, \ldots, n\} \right\}$$

- kleinste mögliche Distanz zwischen zwei Werten der Menge ist $\frac{1}{n(n-1)}$.

Der größtmögliche Durchschnittsgrad

Algorithmus 5 : DensestSubgraph mit MinCut und binärer Suche

Input : Graph $G = (V, E)$
Output : Eine Menge von k Knoten von G

Initialisiere $l := 0$, $r := m$ und $U := \emptyset$;
while $r - l \geq \frac{1}{n(n-1)}$ **do**
    $\gamma := \frac{l+r}{2}$;
    Konstruiere Fluss-Netzwerk (V', E', u_γ);
    Finde Minimum-Schnitt (S, T) des Fluss-Netzwerks;
    if $S = \{s\}$ **then**
        $r := \gamma$
    else
        $l := \gamma$;
        $U := S - \{s\}$
return U

Der größtmögliche Durchschnittsgrad

- Iteration wird $\lceil \log((m+1)n(n-1)) \rceil = \mathcal{O}(\log n)$-mal ausgeführt
- In jeder Iteration MinCut-Berechnung
  mit Push-Relabel-Algorithmus (Goldberg/Tarjan) in $\mathcal{O}(mn \log \frac{n^2}{m})$ für ein Netzwerk mit n Knoten und m Kanten
  (Wir haben zwar $n+2$ Knoten und $2m+2n$ Kanten, das ändert asymptotisch aber nichts.)

$\Rightarrow$ Gesamtkomplexität: $\mathcal{O}\left(mn(\log n)(\log \frac{n^2}{m})\right)$

- mit parametrischen MaxFlow-Algorithmen: $\mathcal{O}\left(mn \log \frac{n^2}{m}\right)$

Durchschnittlicher Grad in gerichteten Graphen

- Es ist nicht unbedingt offensichtlich, wie Dichte (i.S.d. Durchschnittsgrads) auf gerichtete Graphen zu übertragen ist.
- Durchschnittlicher Eingangs- und Ausgangsgrad sind gleich.

$\Rightarrow$ keine orientierungsabhängigen Maße

- Hubs & Authorities: Für gerichteten Graph $G = (V, E)$ und nichtleere (nicht unbedingt disjunkte) Knotenmengen $S, T \subseteq V$ sei $E(S, T)$ die Menge der Kanten, die von einem Knoten in S zu einem Knoten in T gehen:

$$E(S, T) = \{(u, v) : u \in S \text{ und } v \in T\}$$

Definiere durchschnittlichen gerichteten Grad des Paars (S, T) als (Kannan/Vinay)

$$\overline{\deg}_G(S, T) = \frac{|E(S, T)|}{\sqrt{|S| \cdot |T|}}$$

Durchschnittlicher gerichteter Grad

- S ist dann die Menge der Hubs,
  T ist die Menge der Authorities
- Wenn $S = T$, dann kommt genau der konventionelle Durchschnittsgrad heraus.
- Durchschnittsgrad-Maximum eines gerichteten Graphen $G = (V, E)$:

$$\gamma^* = \max_{\substack{S, T \subseteq V \\ S \neq \emptyset, T \neq \emptyset}} \left\{ \overline{\deg}_G(S, T) \right\}$$

- Kann mit Linear Programming in Polynomialzeit gelöst werden (Charikar, 2000).

Dense ℓ-Subgraph

- Graph mit Durchschnittsgrad $\overline{\deg}(G)$ muss nicht unbedingt einen echten Teilgraphen mit gleichem Durchschnittsgrad enthalten.
- Maximum des Durchschnittsgrads eines Teilgraphs auf ℓ Knoten:

$$\gamma^*(G,\ell) = \max\left\{\overline{\deg}(G[U]) : U \subseteq V \text{ und } |U| = \ell\right\}$$

Problem

Problem: ***Dense-ℓ-Subgraph***
Eingabe: *Graph G, Parameter $\ell \in \mathbb{N}$*
Frage: *Eine Knotenmenge der Kardinalität ℓ mit maximalem induzierten Durchschnittsgrad*

Dense ℓ-Subgraph

- Optimierungsproblem **Dense ℓ-Subgraph** ist $\mathcal{NP}$-hart (Instanz $(G, \ell, \ell - 1)$ des zugehörigen Entscheidungsproblems entspricht der Suche nach einer Clique der Größe ℓ in G).

$\Rightarrow$ Wie sieht es mit Approximation aus?

Greedy-Approximation für **Dense ℓ-Subgraph**

Algorithmus 6 : Approximation eines ℓ-Subgraph mit hohem $\overline{\deg}$

Input : Graph $G = (V, E)$ und
gerader Parameter $\ell \in \mathbb{N}$ (mit $|V| \geq \ell$)

Output : Menge von ℓ Knoten von G

Sortiere die Knoten in absteigender Reihenfolge ihrer Grade;
Sei H die Menge von $\frac{\ell}{2}$ Knoten von höchstem Grad;
Berechne $N_H(v) = |N(v) \cap H|$ für alle Knoten $v \in V \setminus H$;
Sortiere die Knoten in $V \setminus H$ in absteigender Reihenfolge der N_H-Werte;
Sei R die Menge von $\frac{\ell}{2}$ Knoten von $V \setminus H$ mit den höchsten N_H-Werten;
return $H \cup R$

Greedy-Approximation für **Dense ℓ-Subgraph**

Satz

Sei G ein Graph auf n Knoten und sei $\ell \in \mathbb{N}$ eine gerade natürliche Zahl mit $\ell \leq n$.
Sei $A(G, \ell)$ der Durchschnittsgrad des induzierten Teilgraphen, der vom vorstehenden Algorithmus ausgegeben wird.
Dann gilt:

$$\gamma^*(G, \ell) \leq \frac{2n}{\ell} \cdot A(G, \ell)$$

bzw.

$$A(G, \ell) \geq \frac{\ell}{2n} \cdot \gamma^*(G, \ell)$$

Greedy-Approximation für **Dense ℓ-Subgraph**

Beweis.

- Für Knotenteilmengen $U, U' \subseteq V$ sei $E(U, U')$ die Menge der Kanten mit einem Endpunkt in U und einem Endpunkt in U'.
- Sei $m_U = |E(G[U])|$
- Sei $\deg_H$ der Durchschnittsgrad der $\frac{\ell}{2}$ Knoten von G mit höchstem Grad bezüglich G. Es gilt: $\deg_H \geq \gamma^*(G, \ell)$.
- Man erhält für die Anzahl der Kanten zwischen H und dem Rest $V \setminus H$:

$$|E(H, V \setminus H)| = \deg_H \cdot |H| - 2m_H \geq \frac{\deg_H \cdot \ell}{2} - 2m_H \geq 0$$

Greedy-Approximation für **Dense ℓ-Subgraph**

Beweis.

- Weil der Algorithmus greedy (gierig) arbeitet, muss der Anteil der Kanten nach R (i.Vgl. zu $V \setminus H$) mindestens so groß sein, wie der Anteil der Knoten:

$$\frac{|E(H,R)|}{|E(H,V\setminus H)|} \geq \frac{|R|}{|V\setminus H|} = \frac{\ell/2}{n-\ell/2} = \frac{\ell}{2n-\ell} > \frac{\ell}{2n}$$

- Also ist die Gesamtzahl der Kanten in $G[H \cup R]$ mindestens

$$\left(\frac{\deg_H \cdot \ell}{2} - 2m_H\right) \cdot \frac{\ell}{2n} + m_H \geq \frac{\deg_H \cdot \ell^2}{4n}$$

Approximation von **Dense ℓ-Subgraph**

- Die Approximationsgüte wird umso besser, je größer ℓ im Vergleich zu n ist.

- Es gibt andere Approximationsverfahren mit Güte $\mathcal{O}(\frac{n}{\ell})$, z.B. durch rekursives Löschen von Knoten mit kleinstem Grad.

Übersicht

Zusammenhang in Graphen / Netzwerken

- beschäftigt sich mit der Stärke der Verbindung zwischen zwei Knoten in Bezug auf die Anzahl knoten- bzw. kantendisjunkter Wege
- "Eine Kette ist nur so stark wie ihr schwächstes Glied."

$\Rightarrow$ Wir suchen nach den schwächsten Elementen, die beim Entfernen die Verbindung zerstören.

Definition

Ein ungerichteter Graph heißt zusammenhängend, wenn es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt.

Ein maximaler zusammenhängender induzierter Teilgraph wird als Zusammenhangskomponente bezeichnet.

Knoten-Zusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt k-knotenzusammenhängend, falls $|V| > k$ und für jede echte Knotenteilmenge $X \subset V$ mit $|X| < k$ der Graph $G - X$ zusammenhängend ist.
Der Knotenzusammenhang $\kappa(G)$ des Graphen G ist die größte natürliche Zahl k, für die G k-knotenzusammenhängend ist.

Bemerkungen:

- Jeder nicht-leere Graph ist 0-knotenzusammenhängend, da es keine Teilmenge X mit $|X| < 0$ gibt.
- Obwohl es wünschenswert wäre, dass die Bezeichnung "1-knotenzusammenhängend" gleichzusetzen ist mit der Bezeichnung "zusammenhängend", wird üblicherweise der Graph bestehend aus nur einem einzelnen Knoten zwar als zusammenhängend, aber nicht 1-zusammenhängend bezeichnet.

Kanten-Zusammenhang und k-Komponenten

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt k-kantenzusammenhängend, falls $|V| \geq 2$ und für jede Kantenteilmenge $Y \subseteq E$ mit $|Y| < k$ der Graph $G - Y$ zusammenhängend ist.

Der Kantenzusammenhang $\lambda(G)$ des Graphen G ist die größte natürliche Zahl k, für die G k-kantenzusammenhängend ist.

Der Kantenzusammenhang eines unzusammenhängenden Graphen sowie des Graphen bestehend aus einem einzelnen Knoten ist 0.

Definition

Die maximalen k-fach knoten-/kanten-zusammenhängenden Teilgraphen werden als k-Knoten-/Kanten-Zusammenhangskomponenten bezeichnet.

Zusammenhang in gerichteten Graphen

Definition

Ein gerichteter Graph ist stark zusammenhängend, wenn es für jeden Knoten einen gerichteten Pfad zu jedem anderen Knoten gibt.

Ein maximaler stark zusammenhängender induzierter Teilgraph wird als starke Zusammenhangskomponente bezeichnet.

Knoten- und Kantenzusammenhang können auf gerichtete Graphen übertragen werden, indem man in der jeweiligen Definition fordert, dass $G - X$ bzw. $G - Y$ *stark zusammenhängend* ist.

Separatoren

Definition

Sei $G = (V, E)$ ein ungerichteter Graph.
Eine Knotenteilmenge $C \subset V$ heißt Knoten-Separator, wenn die Anzahl der Zusammenhangskomponenten in $G - C$ größer als in G ist.

Falls zwei Knoten s und t zwar in G in der gleichen Zusammenhangskomponente sind, aber nicht in $G - C$, dann bezeichnet man C als s-t-Knoten-Separator.

Kanten-Separatoren und s-t-Kanten-Separatoren sind analog definiert.

s-t-Separatoren können auch auf gerichtete Graphen übertragen werden: eine Knoten- bzw. Kantenmenge ist dann ein s-t-Separator, wenn es keinen gerichteten Pfad mehr von s nach t gibt, nachdem die Menge aus dem Graph entfernt wurde.

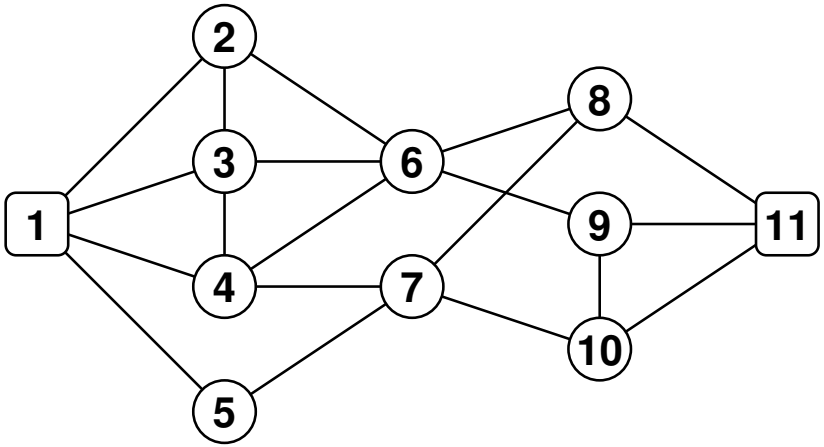
Disjunkte Pfade

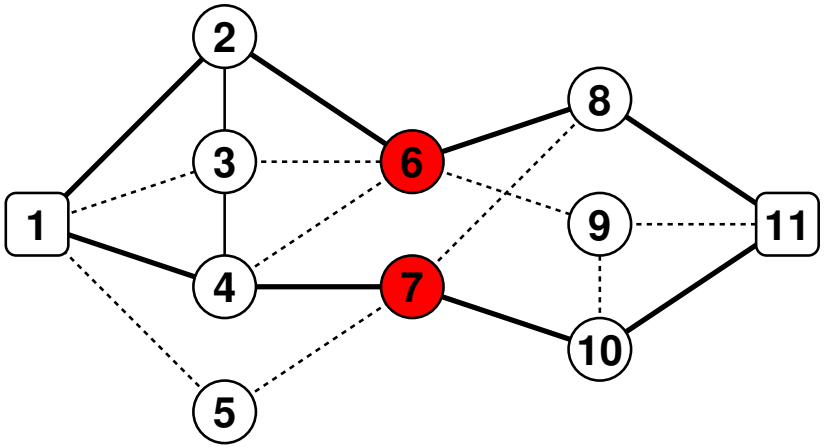
Definition

Zwei (gerichtete oder ungerichtete) Pfade von s nach t werden als knotendisjunkte s-t-Pfade bezeichnet, wenn sie keinen Knoten außer s und t gemeinsam haben.

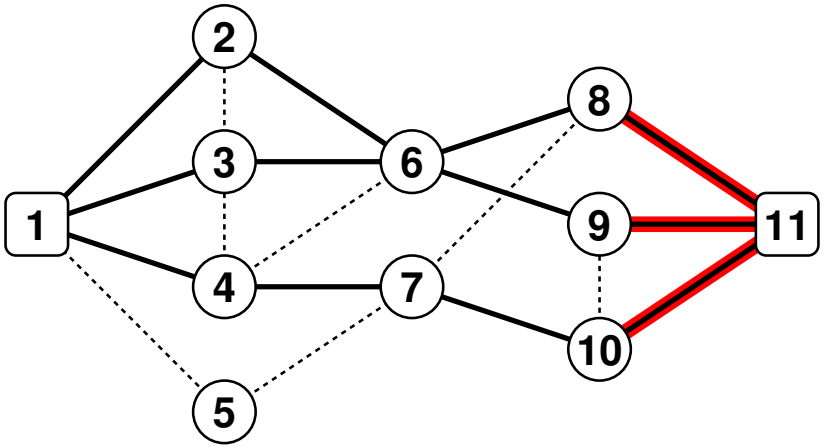
Zwei Pfade werden als kantendisjunkte Pfade bezeichnet, wenn sie keine Kante gemeinsam haben.

Disjunkte s-t-Pfade





2 knotendisjunkte 1-11-Pfade



3 kantendisjunkte 1-11-Pfade

Lokaler Zusammenhang

Definition

Für zwei Knoten s und t eines Graphen G ist der lokale (Knoten-)Zusammenhang $\kappa_G(s,t)$ definiert als die minimale Anzahl von Knoten, die entfernt werden müssen, damit es keinen Weg mehr von s nach t gibt.

Für den Fall, dass zwischen s und t eine Kante existiert, können sie nicht durch das Löschen von Knoten separiert werden. Deshalb wird der lokale (Knoten-)Zusammenhang in diesem Fall $\kappa_G(s,t) = n-1$ definiert. (Anderenfalls wäre höchstens $\kappa_G(s,t) = n-2$ möglich.)

Der lokale Kanten-Zusammenhang zweier Knoten s und t ist entsprechend definiert als die minimale Anzahl von Kanten, die entfernt werden müssen, damit es keinen Weg mehr von s nach t gibt.

Lokaler Zusammenhang

Hinweis:
Für ungerichtete Graphen gilt $\kappa_G(s,t) = \kappa_G(t,s)$ und $\lambda_G(s,t) = \lambda_G(t,s)$, was für gerichtete Graphen im Allgemeinen nicht gilt.

Zweifachzusammenhang

Definition

Ein Artikulationsknoten ist ein Knoten, der beim Entfernen aus dem Graphen die Anzahl der Zusammenhangskomponenten erhöht.
Eine Brücke ist eine Kante, die beim Entfernen aus dem Graphen die Anzahl der Zusammenhangskomponenten erhöht.
Eine Zweifachzusammenhangskomponente ist ein maximaler 2-fach (knoten-)zusammenhängender Teilgraph.
Ein Block ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält, d.h. die Menge aller Blocks eines Graphen besteht aus den isolierten Knoten, den Brücken, sowie den Zweifachzusammenhangskomponenten.

Block-Graph und CutPoint-Graph

Definition

Der Block-Graph $B(G)$ eines Graphen G hat jeweils einen Knoten für jeden Block von G, wobei zwei Knoten des Block-Graphen adjazent sind, wenn die entsprechenden Blöcke in G einen (Artikulations-)Knoten gemeinsam haben.
Der CutPoint-Graph $C(G)$ eines Graphen G hat jeweils einen Knoten für jeden Artikulationsknoten von G, wobei zwei Knoten des CutPoint-Graphen adjazent sind, wenn die entsprechenden Artikulationsknoten in G zum gleichen Block gehören.

Satz (Harary)

Für jeden Graphen gilt:

$$B(B(G)) = C(G) \quad \textit{und} \quad B(C(G)) = C(B(G))$$

Block-CutPoint-Graph

Definition

Der Block-CutPoint-Graph eines Graphen G ist der bipartite Graph, dessen Knotenmenge aus je einem Knoten für jeden Artikulationsknoten von G und je einem Knoten für jeden Block von G besteht, wobei ein CutVertex-Knoten mit einem Block-Knoten genau dann durch eine Kante verbunden ist, wenn der Artikulationsknoten zu dem entsprechenden Block gehört.

Satz (Harary & Prins)

Der Block-CutPoint-Graph eines zusammenhängenden Graphen ist ein Baum.

Fundamentale Ungleichung

Satz

Für jeden nicht-trivialen Graphen G gilt:

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

"nicht-trivial" soll den Graph ohne Knoten und den Graph auf einem Knoten ausschließen, hier kommt es auf die genaue Definition von κ und λ an.

Beweis.

- Die inzidenten Kanten eines Knotens v mit $\deg(v) = \delta(G)$ bilden einen Kanten-Separator.

$\Rightarrow \lambda(G) \leq \delta(G)$

Fundamentale Ungleichung

Beweis.

- Sei $G = (V, E)$ ein Graph mit mindestens zwei Knoten.
  Betrachte minimalen Kanten-Separator, der die Knotenmenge in S und $\bar{S} = V \setminus S$ partitioniert.
- Für den Fall, dass alle Kanten zwischen S und $\bar{S}$ vorhanden sind, gilt $\lambda(G) = |S| \cdot |\bar{S}| \geq n - 1$ (und $\kappa(G)$ kann nicht größer als $n - 1$ sein).
- Anderenfalls existieren Knoten $x \in S$ und $y \in \bar{S}$ mit $\{x, y\} \notin E$, wobei die Nachbarn von x in $\bar{S}$ zusammen mit allen Knoten aus $S \setminus \{x\}$, die Nachbarn in $\bar{S}$ haben, einen Knoten-Separator bilden.
  Dieser Knoten-Separator ist höchstens so groß wie die Anzahl der Kanten von S nach $\bar{S}$, und er separiert mindestens x und y.

Das n-Chain / n-Arc Theorem

Satz (Menger, 1927)

Seien P und Q Teilmengen der Knoten eines ungerichteten Graphen. Dann ist die maximale Anzahl knotendisjunkter Pfade, die Knoten von P mit Knoten von Q verbinden, gleich der minimalen Kardinalität einer Knotenmenge, die alle Pfade zwischen Knoten in P und Knoten in Q überschneidet bzw. unterbricht.

Der Satz von Menger

Satz ("Satz von Menger")

Seien s und t zwei Knoten eines ungerichteten Graphen. Wenn s und t nicht adjazent sind, dann ist die maximale Anzahl knotendisjunkter s-t-Pfade gleich der minimalen Kardinalität eines s-t-Knoten-Separators.

Satz (Kantenversion)

Die maximale Anzahl kantendisjunkter s-t-Pfade ist gleich der minimalen Kardinalität eines s-t-Kanten-Separators.

(Ford/Fulkerson, Dantzig/Fulkerson, Elias/Feinstein/Shannon, 1956)

Eng verwandt: MaxFlow-MinCut-Theorem
(Kantenversion von Menger's Theorem kann als Spezialfall gesehen werden, wo alle Kantengewichte den gleichen Wert haben.)

Whitney's Theorem

Satz (Whitney, 1932)

Sei G ein nicht-trivialer Graph und k eine natürliche Zahl.
G ist genau dann k-(knoten-)zusammenhängend, wenn alle Paare verschiedener Knoten (s, t) durch k knotendisjunkte s-t-Pfade verbunden werden können.

Schwierig bei der Herleitung ist nur, dass der Satz von Menger fordert, dass die Knoten nicht adjazent sind.
Da diese Bedingung bei der Kantenversion nicht auftritt, folgt aus dieser sofort:

Satz

Sei G ein nicht-trivialer Graph und k eine natürliche Zahl.
G ist genau dann k-kanten-zusammenhängend, wenn alle Paare verschiedener Knoten (s, t) durch k kantendisjunkte s-t-Pfade verbunden werden können.

Gemischter Knoten-/Kanten-Zusammenhang

Definition

Ein Paar (k, l) heißt *Zusammenhangspaar* zweier Knoten s und t eines Graphen, falls eine Menge aus k Knoten und l Kanten existiert, die jeden Weg zwischen s und t beim Entfernen zerstört, aber es keine solche Menge bestehend aus $k-1$ Knoten und l Kanten oder k Knoten und $l-1$ Kanten gibt.

Satz (Beineke & Harary, 1967)

Wenn (k, l) ein Zusammenhangspaar zweier Knoten s und t in einem Graphen ist, dann gibt es $k + l$ kantendisjunkte Pfade von s nach t, von denen k knotendisjunkte s-t-Pfade sind.

Einfache Schranken

Satz

Der maximale (Knoten-/Kanten-)Zusammenhang in einem Graphen mit n Knoten und m Kanten ist

$$\begin{array}{rl} \left\lfloor \frac{2m}{n} \right\rfloor : & \text{falls } m \geq n-1 \\ 0 : & \text{sonst} \end{array}$$

Der minimale (Knoten-/Kanten-)Zusammenhang in einem Graphen mit n Knoten und m Kanten ist

$$\begin{array}{rl} m - \binom{n-1}{2} : & \text{falls } \binom{n-1}{2} < m \leq \binom{n}{2} \\ 0 : & \text{sonst} \end{array}$$

Für jeden Graphen G mit $\delta(G) \geq \left\lfloor \frac{n}{2} \right\rfloor$ gilt: $\lambda(G) = \delta(G)$.

Überlappung von k-Knoten-Komponenten

Die offensichtliche Tatsachen, dass

- zwei verschiedene Zusammenhangskomponenten keinen Knoten gemeinsam haben können und
- zwei verschiedene Blöcke höchstens einen Knoten gemeinsamen haben können,

lässt sich wie folgt verallgemeinern:

Satz

Zwei verschiedene k-(Knoten-)Komponenten haben höchstens $k-1$ Knoten gemeinsam.

Nicht-Überlappung von k-Kanten-Komponenten

Satz (Matula, 1968)

Für jede natürliche Zahl k sind die k-Kanten-Komponenten eines Graphen knotendisjunkt.

Nicht-Überlappung von k-Kanten-Komponenten

Beweis.

- Betrachte (überlappende) Zerlegung $\tilde{G} = G_1 \cup G_2 \cup \ldots \cup G_t$ eines zusammenhängenden Teilgraphen $\tilde{G}$ von G.
- Sei $C = (A, \bar{A})$ ein minimaler Kanten-Schnitt von $\tilde{G}$.
- Annahme: $\tilde{G}$ hat mindestens zwei Knoten (sonst trivial)
- Für jeden Teilgraph G_i, der eine bestimmte Kante e des MinCuts C enthält, enthält C auch einen Schnitt für G_i. Ansonsten wäre dieser Teil überflüssig da alle Knoten in $G_i - C$ (und damit auch in $\tilde{G} - C$) noch zusammenhängen, und das würde der Minimum-Bedingung des Schnitts widersprechen.

$\Rightarrow \exists G_i: \quad \lambda(\tilde{G}) = |C| \geq \lambda(G_i)$
(denn jede Kante aus C gehört zu mindestens einer k-Kanten-Komponente G_i)

$\Rightarrow \lambda(\tilde{G}) \geq \min_{1 \leq i \leq t}\{\lambda(G_i)\}$

Satz von Mader

Obwohl aus der fundamentalen Ungleichung $\kappa(G) \leq \lambda(G) \leq \delta(G)$ folgt, dass k-Knoten-/Kanten-Zusammenhang einen Minimalgrad $\geq k$ impliziert, ist das Gegenteil nicht unbedingt der Fall.
Ein hoher Durchschnittsgrad impliziert aber die Existenz eines relativ gut zusammenhängenden Teilgraphen:

Satz (Mader, 1972)

Jeder Graph mit Durchschnittsgrad mindestens $4k$ enthält einen k-zusammenhängenden Teilgraph.

Kanten-Schnitte minimalen Gewichts (Minimum Cuts)

- ungerichteter gewichteter Graph $G = (V, E)$
- zwei disjunkte Knotenteilmengen $X, Y \subseteq V$, $X \cap Y = \emptyset$
- Gewichtssumme der Kanten von einem Knoten in X zu einem Knoten in Y: $w(X, Y)$ (für gerichtete Graphen analog)

Fakt

Ein Kanten-Schnitt minimalen Gewichts in einem zusammenhängenden Graphen mit echt positiven Kantengewichten induziert einen zusammenhängenden Teilgraphen.

Minimum Cuts

Lemma

Sei $(S, V \setminus S)$ ein Minimum Cut in $G = (V, E)$. Dann gilt für jede nicht-leere Teilmenge T von S:

$$w(T, S \setminus T) \geq \frac{\lambda}{2}$$

Beweis.

- Annahme: $w(T, S \setminus T) < \frac{\lambda}{2}$
- $w(T, V \setminus S) + w(S \setminus T, V \setminus S) = \lambda$
- o.B.d.A.: $w(T, V \setminus S) \leq \frac{\lambda}{2}$
  (sonst vertausche T und $S \setminus T$)

$\Rightarrow$ $w(T, V \setminus T) = w(T, S \setminus T) + w(T, V \setminus S) < \lambda$ (Widerspruch)

□

Minimum Cuts

Notation:

- $\bar{X} = V \setminus X$
- Im Folgenden werden wir einen Schnitt $(X, \bar{X})$ oft einfach nur mit X bezeichnen.

Lemma

Seien $(A, \bar{A})$ und $(B, \bar{B})$ mit $A \neq B$ zwei Minimum Cuts in $G = (V, E)$, so dass $T = A \cup B$ auch ein Minimum Cut in G ist. Dann gilt:

$$w(A, \bar{T}) = w(B, \bar{T}) = w(A \setminus B, B) = w(A, B \setminus A) = \frac{\lambda}{2}$$

Minimum Cuts

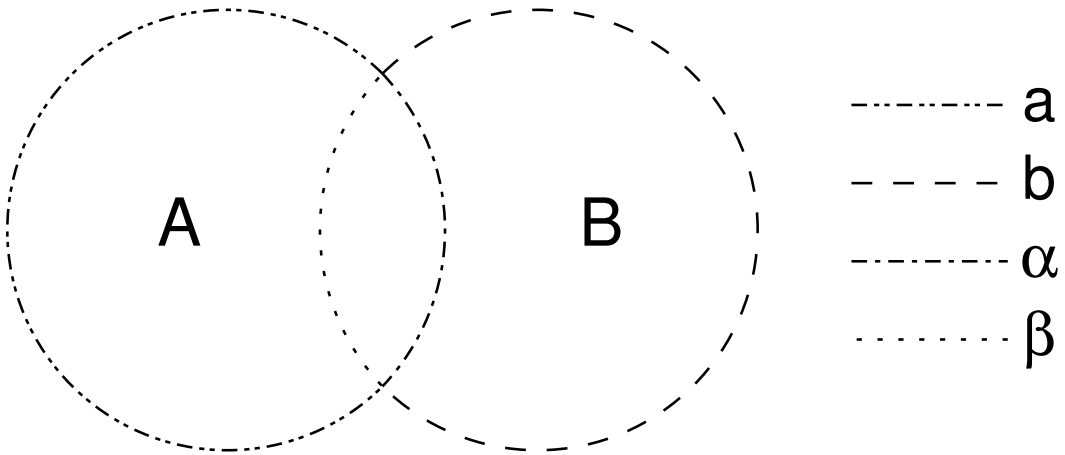


Abbildung: Schnitt zweier Minimum Cuts A und B

Minimum Cuts

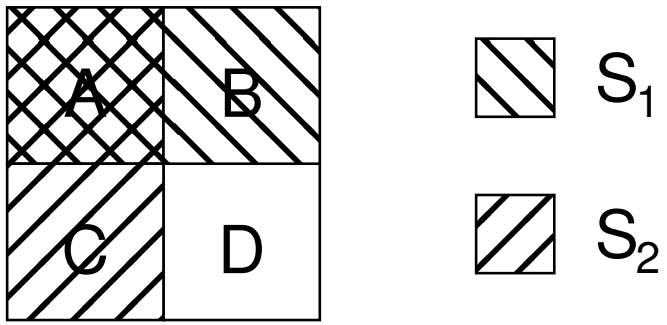
Beweis.

- Sei $a = w\left(A, \bar{T}\right)$,
  $b = w\left(B, \bar{T}\right)$,
  $\alpha = w\left(A, B \setminus A\right)$ und
  $\beta = w\left(B, A \setminus B\right)$.

$\Rightarrow$ $w\left(A, \bar{A}\right) = a + \alpha = \lambda$,
$w\left(B, \bar{B}\right) = b + \beta = \lambda$
$w\left(T, \bar{T}\right) = a + b = \lambda$

- Es gilt auch:
  $w\left(A \setminus B, B \cup \bar{T}\right) = a + \beta \geq \lambda$ und $w\left(B \setminus A, A \cup \bar{T}\right) = b + \alpha \geq \lambda$.
- Dieses (Un-)Gleichungssystem hat nur eine Lösung:
  $a = \alpha = b = \beta = \frac{\lambda}{2}$.

Minimum Cuts

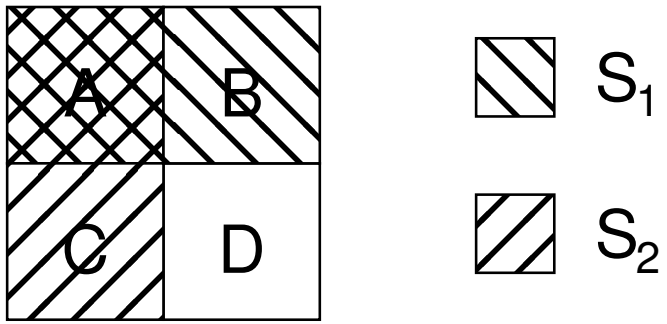


Definition

Ein Paar $\langle S_1, S_2 \rangle$ heißt Crossing Cut, falls S_1, S_2 Minimum Cuts sind und keine der folgenden Mengen leer ist:

- $A = S_1 \cap S_2$,
- $B = S_1 \setminus S_2$,
- $C = S_2 \setminus S_1$
- $D = \bar{S}_1 \cap \bar{S}_2$

Crossing Cuts



Lemma

Seien $\langle S_1, S_2 \rangle$ Crossing Cuts und seien Mengen wie folgt definiert: $A = S_1 \cap S_2$, $B = S_1 \setminus S_2$, $C = S_2 \setminus S_1$ and $D = \bar{S}_1 \cap \bar{S}_2$. Dann gilt:

1. *A, B, C und D sind Minimum Cuts.*
2. $w(A, D) = w(B, C) = 0$
3. $w(A, B) = w(B, D) = w(D, C) = w(C, A) = \frac{\lambda}{2}$

Crossing Cuts

Beweis.

- Da S_1 und S_2 Minimum Cuts sind, gilt:
  - $w\left(S_1, \bar{S}_1\right) = w(A,C) + w(A,D) + w(B,C) + w(B,D) = \lambda$
  - $w\left(S_2, \bar{S}_2\right) = w(A,B) + w(A,D) + w(B,C) + w(C,D) = \lambda$

$\Rightarrow$ $w(A,B) + w(A,C) + 2w(A,D) + 2w(B,C) + w(B,D) + w(C,D) = 2\lambda$

- Da es keinen Schnitt mit Kapazität $< \lambda$ gibt, gilt:

$$\begin{array}{lcll} w\left(A, \bar{A}\right) & = & w(A,B) + w(A,C) + w(A,D) & \geq \lambda \\ w\left(B, \bar{B}\right) & = & w(A,B) + w(B,C) + w(B,D) & \geq \lambda \\ w\left(C, \bar{C}\right) & = & w(A,C) + w(B,C) + w(C,D) & \geq \lambda \\ w\left(D, \bar{D}\right) & = & w(A,D) + w(B,D) + w(C,D) & \geq \lambda \end{array}$$

$\Rightarrow$ $2[w(A,B) + w(A,C) + w(A,D) + w(B,C) + w(B,D) + w(C,D)] \geq 4\lambda$

$\Rightarrow$ $w(A,D) = w(B,C) = 0$ (keine Diagonalkanten)

Crossing Cuts

Beweis.

- Die waagerechte Linie in der Abbildung entspricht dem Minimum Cut $(S_1, \bar{S}_1) = (A \cup B, C \cup D) = \lambda$, die senkrechte entspricht $(S_2, \bar{S}_2) = (A \cup C, B \cup D) = \lambda$.
- Analogie: Länge der Kanten entspricht Kapazität der geschnittenen Kanten
- Annahme: die waagerechte und senkrechte Linie schneiden sich nicht genau in der Mitte

$\Rightarrow$ Dann definiert eine der Teilmengen $X = A$, B, C oder D einen Schnitt $w(X, \bar{X}) < \lambda$ (Widerspruch)

$\Rightarrow$ $w(A, B) = w(B, D) = w(D, C) = w(C, A) = \frac{\lambda}{2}$ und $w(A, \bar{A}) = w(B, \bar{B}) = w(C, \bar{C}) = w(D, \bar{D}) = \lambda$

□

Zirkuläre Partitionen

Crossing Cuts in $G = (V, E)$ partitionieren V in vier Teile
Allgemeiner:

Definition

Eine zirkuläre Partition ist eine Partition von V in $k \geq 3$ disjunkte Mengen $V_1, V_2, \ldots, V_k$, so dass

- $w(V_i, V_j) = \begin{cases} \lambda/2 & \text{falls } |i-j| = 1 \mod k \\ 0 & \textit{sonst} \end{cases}$
- Falls S ein Minimum Cut ist, dann ist
  - S oder $\bar{S}$ eine echte Teilmenge einer Menge V_i oder
  - die zirkuläre Partition ist eine Verfeinerung der Partition, die durch den Minimum Cut S definiert wird.
    (S ist die Vereinigung einiger Mengen der zirkulären Partition.)

Zirkuläre Partitionen

- Seien $V_1, V_2, \ldots, V_k$ die disjunkten Mengen einer zirkulären Partitionierung.
- Dann ist für alle a, b mit $1 \leq a \leq b < k$ die Menge $S = \bigcup_{i=a}^{b} V_i$ (zusammen mit $\bar{S}$) ein Minimum Cut.
- Diese Minimum Cuts bezeichnen wir als Circular Partition Cuts.
- Insbesondere ist jedes V_i $(1 \leq i \leq k)$ ein Minimum Cut (erster Punkt der vorstehenden Definition).
- Betrachte einen Minimum Cut S, so dass weder S noch $\bar{S}$ in einer Menge der zirkulären Partition enthalten ist. Da S zusammenhängend ist, ist S oder $\bar{S}$ gleich $\bigcup_{i=a}^{b} V_i$ für ein Paar a, b mit $1 \leq a < b < k$
- Für alle V_i einer zirkulären Partitionierung existiert kein Minimum Cut S, so dass $\langle V_i, S \rangle$ ein Crossing Cut ist (zweiter Punkt der vorstehenden Definition).

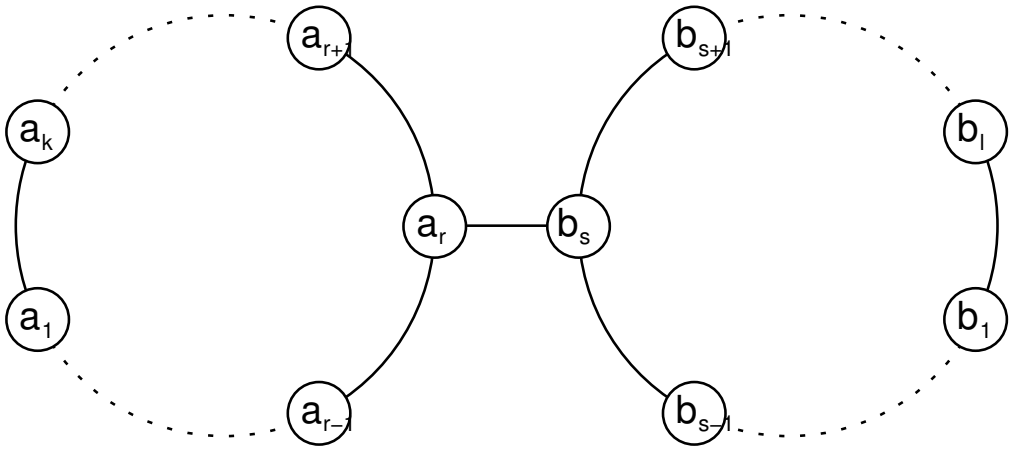
Kompatibilität zirkulärer Partitionen

Definition

Zwei verschiedene zirkuläre Partitionen $P = \{U_1, \ldots, U_k\}$ und $Q = \{V_1, \ldots, V_l\}$ sind kompatibel, wenn es eindeutige Zahlen r und s $(1 \leq r, s \leq k)$ gibt, so dass $\forall i \neq r : U_i \subseteq V_s$ und $\forall j \neq s : V_j \subseteq U_r$.

Kompatibilität zirkulärer Partitionen

Beispiel:



$$P = \{\{a_1\}, \ldots, \{a_{r-1}\}, \{a_r, b_1, \ldots b_l\}, \{a_{r+1}\}, \ldots \{a_k\}\}$$

$$Q = \{\{b_1\}, \ldots, \{b_{s-1}\}, \{b_s, a_1, \ldots a_k\}, \{b_{s+1}\}, \ldots \{b_l\}\}$$

Lemma

Alle verschiedenen zirkulären Partitionen sind paarweise kompatibel.

Kompatibilität zirkulärer Partitionen

Beweis.

- Betrachte zwei zirkuläre Partitionen P und Q in $G = (V, E)$.
- Alle Mengen der Partitionen sind Minimum Cuts.
- Annahme: eine Menge $S \in P$ ist die Vereinigung von mindestens zwei, aber nicht von allen Mengen von Q.
- Genau zwei Mengen $A, B \in Q$, die in S enthalten sind, sind durch mindestens eine Kante zu den Knoten in $V \setminus S$ verbunden.
- Sei T die Menge, die man durch Ersetzen von $A \subset S$ durch ein Element von Q erhält, das zu B verbunden, aber nicht in S enthalten ist.

$\Rightarrow$ Dann ist $\langle S, T \rangle$ ein Crossing Cut (Widerspruch)

$\Rightarrow$ Jede Menge von P oder ihr Komplement ist in einer Menge von Q enthalten.

Kompatibilität zirkulärer Partitionen

Beweis.

- Annahme: zwei Mengen von P sind in zwei verschiedenen Mengen von Q enthalten.
- Da jedes Komplement der verbleibenden Mengen von P nicht in *einer* Menge von Q enthalten sein kann, muss jede übrige Menge von P in einer Teilmenge von Q enthalten sein.

$\Rightarrow$ $P = Q$ (Widerspruch)

- Annahme: alle Mengen von P sind in einer Menge Y von Q

$\Rightarrow$ $Y = V$ (Widerspruch)

- Da die Vereinigung von zwei Komplementen von Mengen aus P gleich V ist und Q mindestens drei Mengen enthält, kann nur ein Komplement in einer Menge von Q enthalten sein.

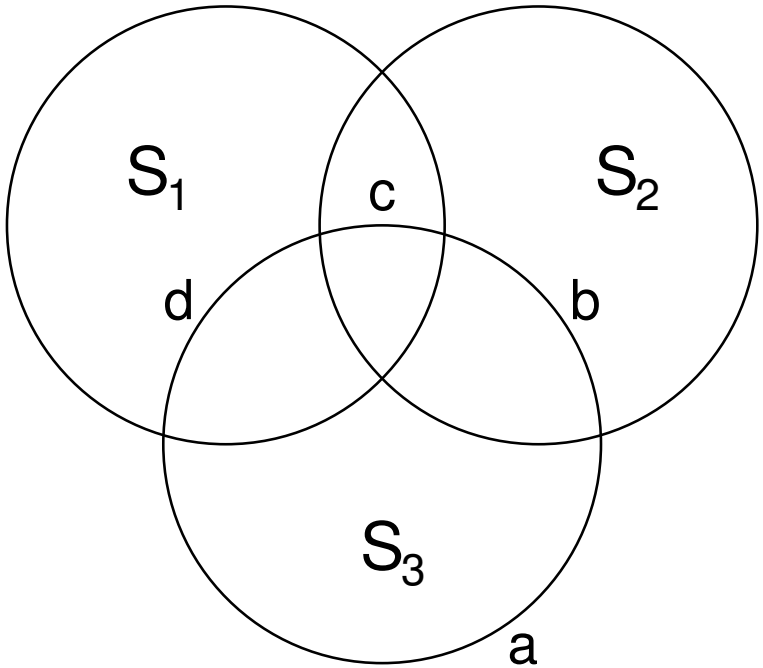
$\Rightarrow$ Es gibt genau eine Menge X in P, die nicht in Y von Q enthalten ist, aber $\bar{X} \subset Y$.

Paarweise Disjunktheit von Crossing Cuts

Lemma

Wenn S_1, S_2 und S_3 paarweise Crossing Cuts sind, dann gilt:

$$S_1 \cap S_2 \cap S_3 = \emptyset$$



Paarweise Disjunktheit von Crossing Cuts

Beweis.

- Annahme: das Lemma ist falsch, also die Schnittmenge ist nicht leer
- Definiere
  - $a = w\left(S_3 \setminus (S_1 \cup S_2), \overline{S_1 \cap S_2 \cap S_3}\right)$
  - $b = w\left((S_2 \cap S_3) \setminus S_1, S_2 \setminus (S_1 \cup S_3)\right)$
  - $c = w\left(S_1 \cap S_2 \cap S_3, (S_1 \cap S_2) \setminus S_3\right)$
  - $d = w\left((S_1 \cap S_3) \setminus S_2, S_1 \setminus (S_2 \cup S_3)\right)$
- Einerseits ist $S_1 \cap S_2$ ein Minimum Cut. $\Rightarrow \quad c \geq \frac{\lambda}{2}$
- Andererseits ist $c + b = c + d = \frac{\lambda}{2}$.

$\Rightarrow$ $b = d = 0$ und $(S_1 \cap S_3) \setminus S_2 = (S_2 \cap S_3) \setminus S_1 = \emptyset$

- Da es keine diagonalen Kanten in Crossing Cuts gibt (siehe früheres Lemma), sind $S_1 \cap S_2 \cap S_3$ und $S_3 \setminus (S_1 \cup S_2)$ nicht verbunden. (Widerspruch)

Weitere Sätze

Satz

In einem Graphen $G = (V, E)$ gibt es für jede einem Crossing Cut entsprechende Partition P von V in 4 disjunkte Mengen eine zirkuläre Partition in G, die eine Verfeinerung von P ist.

Satz

Ein Graph $G = (V, E)$ hat $\mathcal{O}\left(\binom{|V|}{2}\right)$ viele Minimum Cuts (und diese Schranke ist scharf).

Kaktus-Vorbereitung: Laminare Mengen

Definition

Eine Menge $\mathcal{S}$ von Mengen heißt laminar, wenn für jedes Paar von Mengen $S_1, S_2 \in \mathcal{S}$ gilt, dass entweder S_1 und S_2 disjunkt sind, oder eine der beiden Menge die andere enthält.

- Jede laminare Menge $\mathcal{S}$ kann als Baum repräsentiert werden.
- Jeder Knoten repräsentiert eine Menge in $\mathcal{S}$.
- Die Blätter des Baums repräsentieren die Mengen von $\mathcal{S}$, die keine anderen Mengen enthalten.
- Der Vater eines zur Menge T gehörigen Knotens repräsentiert die (eindeutige) kleinste Übermenge von T.
- Die Konstruktion liefert eine Menge von Bäumen, deren Wurzelknoten Mengen repräsentieren, die in keiner anderen Menge von $\mathcal{S}$ enthalten sind.

Kaktus-Vorbereitung: Laminare Mengen

- Hinzufügen eines künstlichen Wurzelknotens, der mit allen eigentlichen Wurzeln verbunden ist, liefert einen Baum.

$\Rightarrow$ Die Knoten eines Baums repräsentieren alle Mengen von $\mathcal{S}$, wobei die Wurzel die zugrundeliegende Gesamtmenge (Vereinigung aller Mengen) repräsentiert.

- Wenn diese Vereinigung n Elemente enthält, kann der Baum höchstens n Blätter bzw. $2n - 1$ Knoten haben.

Kaktus

Definition

Ein zusammenhängender Graph heißt Kaktus, falls jedes Paar von einfachen Kreisen höchstens einen Knoten gemeinsam hat.
Ein Graph bestehend aus einem einzelnen Knoten wird als *trivialer Kaktus* bezeichnet.

Ein Graph ist ein Kaktus genau dann, wenn jede Zweifachzusammenhangskomponente entweder ein einfacher Kreis oder eine einzelne Kante (Brücke) ist.
Hinweis: Oft wird in der Definition auch einfach verlangt, dass jede Kante zu genau einem (knoten-disjunkten) Kreis gehört, wobei eine Doppelkante zwischen zwei Knoten einen Kreis der Länge 2 darstellt.
Man kann einen Kaktus nach dieser Definition erhalten, indem man die Brücken durch Doppelkanten ersetzt.

Kaktus-Repräsentation von Minimum Cuts

- Betrachte
  - einen Graph G,
  - einen ungewichteten Kaktus $\mathcal{R}$ und
  - eine Abbildung φ der Graphknoten in die Menge der Kaktusknoten $\varphi: V(G) \to V(\mathcal{R})$.

- Die Menge $V(\mathcal{R})$ kann einen Knoten x enthalten, der nicht in der Bildmenge der Abbildung φ enthalten ist, für den also $V(G)$ keinen Knoten v mit $\varphi(v) = x$ enthält.
  Ein solcher Knoten wird *leerer Knoten* genannt.

- Für jeden nichttrivialen (ungewichteten) Kaktus $\mathcal{R}$ gilt $\lambda(\mathcal{R}) \leq 2$ bzw. $\lambda(\mathcal{R}) = 2$ (je nach Definition).

Kaktus-Repräsentation von Minimum Cuts

Sei $\mathcal{C}(\mathcal{R})$ die Menge aller Minimum Cuts vom Kaktus $\mathcal{R}$.
D.h. $\{S, V(\mathcal{R}) \setminus S\} \in \mathcal{C}(\mathcal{R})$ gilt genau dann, wenn $E(S, V(\mathcal{R}) \setminus S; R)$ eine Menge von zwei Kanten ist, die zum gleichen Kreis in $\mathcal{R}$ gehören.

Definition

Für eine gegebene Teilmenge $\mathcal{C}' \subseteq \mathcal{C}(G)$ von Minimum Cuts, nennt man ein Paar $(\mathcal{R}, \varphi)$, das aus einem Kaktus $\mathcal{R}$ und einer Knotenabbildung φ besteht, Kaktus-Repräsentation für $\mathcal{C}'$ falls folgende Bedingungen erfüllt sind:

1. Für einen beliebigen Kaktus-MinCut $\{S, V(\mathcal{R}) \setminus S\} \in \mathcal{C}(\mathcal{R})$ gehört der Cut $\{X, \bar{X}\}$ mit $X = \{u \in V(G) \mid \varphi(u) \in S\}$ und $\bar{X} = \{u \in V(G) \mid \varphi(u) \in V(\mathcal{R}) \setminus S\}$ zu $\mathcal{C}'$.
2. Für jeden MinCut $\{X, \bar{X}\} \in \mathcal{C}'$ existiert ein Kaktus-MinCut $\{S, V(\mathcal{R}) \setminus S\} \in \mathcal{C}(\mathcal{R})$ mit $X = \{u \in V(G) \mid \varphi(u) \in S\}$ und $\bar{X} = \{u \in V(G) \mid \varphi(u) \in V(\mathcal{R}) \setminus S\}$.

Kaktus-Repräsentation aller Minimum Cuts

Fallunterscheidung:

1. Graph ohne zirkuläre Partitionen
2. Graph mit genau einer zirkulären Partition
3. Graph mit mehreren zirkulären Partitionen $P_1, \ldots, P_z$

Kaktus-Repräsentation aller Minimum Cuts

1. Fall: Graph ohne zirkuläre Partitionen

- Wenn es Crossing (Minimum) Cuts geben würde, müsste es laut Lemma auch eine zirkuläre Partition geben, die eine Verfeinerung der 4 entsprechenden disjunkten Mengen ist.

$\Rightarrow$ Es kann in diesem Fall keine Crossing Cuts geben.

$\Rightarrow$ Die Minimum Cuts sind laminar.

$\Rightarrow$ Die Minimum Cuts können durch einen Baum T_G repräsentiert werden.

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 1. Fall: Graph ohne zirkuläre Partitionen)
Die Minimum Cuts können durch folgenden Baum T_G repräsentiert werden:

- Betrachte die jeweils kleinere Knotenmenge jedes Minimum Cuts und bezeichne die Menge dieser Knotenmengen mit Λ
  (wähle bei gleicher Kardinalität irgendeine von beiden).
- Repräsentiere jede Menge von Λ durch einen Knoten in T_G.
- Zwei Baumknoten, die zu den MinCut-Mengen A und B im Graphen gehören, sollen genau dann verbunden sein, wenn $A \subset B$ gilt und es keinen MinCut C mit $A \subset C \subset B$ gibt (der echte Obermenge von A und echte Teilmenge von B ist).
- Die Wurzeln der resultierenden Bäume repräsentieren die MinCuts in Λ, die in keiner anderen MinCut-Menge von Λ enthalten sind.
- Hinzufügen eines künstlichen Wurzelknotens und Verbinden mit den Wurzeln aller Bäume resultiert in einem Baum (T_G).

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 1. Fall: Graph ohne zirkuläre Partitionen)

- Definiere Abbildung:
  - Jeder Knoten des Graphen G auf den Knoten des Baums T_G abgebildet, der zu dem MinCut mit kleinster Kardinalität gehört, der diesen Knoten enthält.
  - Jeder nicht zugeordnete Knoten wird der Wurzel zugeordnet.
- Für jeden Minimum Cut S von G werden die Knoten von S einer Menge X von Baumknoten zugeordnet, so dass es eine Kante gibt, die beim Entfernen die Baumknoten X vom Rest des Baums trennt.
- Andererseits zerfällt beim Entfernen einer Kante aus T_G die Menge der Baumknoten so in zwei Teile, dass die Menge der Knoten, die in dem einen Teil zugeordnet werden, die eine Seite eines Minimum Cuts bilden.

$\Rightarrow$ Wenn der Graph keine zirkulären Partitionen enthält, dann ist der Baum T_G der Kaktus C_G des Graphen G und die Anzahl seiner Knoten ist durch $2|V| - 1$ beschränkt.

Kaktus-Repräsentation aller Minimum Cuts

2. Fall: Graph mit genau einer zirkulären Partition $V_1, \ldots, V_k$.

- Die Circular Partition Cuts können durch einen Kreis mit k Knoten repräsentiert werden.
- Die Knoten jedes Partitionsteils V_i $(1 \leq i \leq k)$ werden so durch einen Knoten N_i des Kreises repräsentiert, dass zwei Teile V_i und V_{i+1} durch zwei adjazente Knoten repräsentiert werden.
- Bemerkung: Für jeden Minimum Cut S, der kein Circular Partition Cut ist, ist entweder S oder $\bar{S}$ eine echte Teilmenge eines Teils V_i (folgt direkt aus der Definition).

Kaktus-Repräsentation aller Minimum Cuts

(Fortsetzung 2. Fall: Graph mit genau einer zirkulären Partition)

- Man kann den Baum $T_{(V_i,E)}$ für alle Minimum Cuts konstruieren, die Teilmenge von V_i sind, aber mit der Beschränkung, dass nur die Knoten von V_i diesem Baum zugeordnet werden.
- Die Wurzel von $T_{(V_i,E)}$ entspricht genau der Menge V_i.

$\Rightarrow$ Knoten N_i des Kreises kann mit der Wurzel von $T_{(V_i,E)}$ verschmolzen werden $(\forall i : 1 \leq i \leq k)$.

- Dieser mit allen Bäumen verbundene Kreis ist der Kaktus C_G für G.
- Anzahl Knoten: Summe der Anzahl der Knoten aller Bäume

$\Rightarrow$ wieder beschränkt durch $2|V| - 1$ und wieder Korrespondenz zwischen Minimum Cuts in G und Separation in C_G.

Kaktus-Repräsentation aller Minimum Cuts

3. Fall: Graph mit zirkulären Partitionen $P_1, \ldots, P_z$

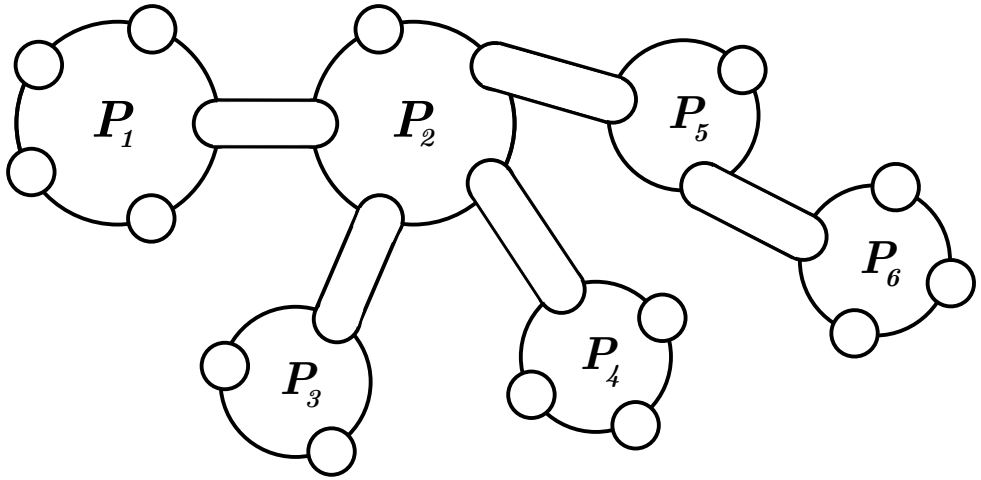
- Betrachte alle zirkulären Partitionen als Menge von Mengen
- Konstruiere den Kaktus, der die Circular Partition Cuts repräsentiert:
  - Die Knoten jeder Menge $F \in \mathcal{F}_{P_1 \cup \ldots \cup P_z}$ werden einem Knoten zugeordnet.
  - Zwei Knoten sind verbunden, wenn für ihre Mengen F_1 und F_2 gilt: $w(F_1, F_2) > 0$.

  $\Rightarrow$ Jede zirkuläre Partition erzeugt einen Kreis in C_G.
  - Da alle zirkulären Partitionen paarweise kompatibel sind, sind die Kreise durch Kanten verbunden, die nicht Teil eines Kreises sind.
  - Der Kaktus C_G ist jetzt ein baumartiger Graph.

Kaktus-Repräsentation aller Minimum Cuts

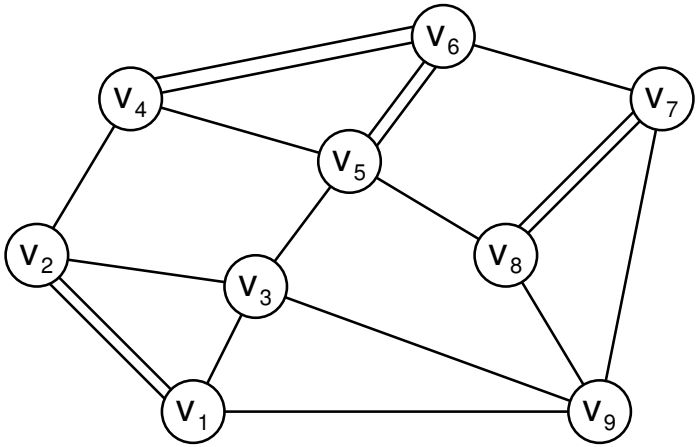
(Fortsetzung 3. Fall: Graph mit zirkulären Partitionen $P_1, \ldots, P_z$)

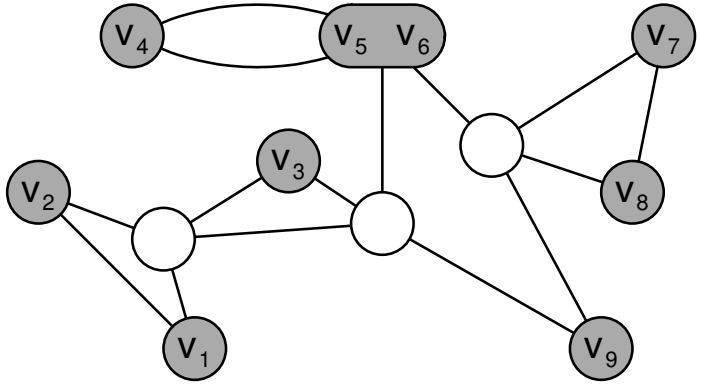
- Bsp.: Kaktus für die Circular Partition Cuts von 6 Circular Partitions



- Repräsentiere die Minimum Cuts, die nicht Teil einer zirkulären Partition sind (analog zum 2. Fall)
- Man erhält den Kaktus T_C von G.
- Die Anzahl Knoten ist wieder beschränkt durch $2|V| - 1$

Beispiel für Kaktus-Repräsentation





Cycle-type normal cactus representations

Definition

In einem Kaktus $\mathcal{R}$ nennen wir einen Kreis mit h Knoten einen h-Kreis. Einen Knoten, der zu genau k Kreisen gehört, nennt man k-Verzweigungsknoten.

Eine Kaktus-Repräsentation heißt *normal*, wenn sie keinen leeren 2-Verzweigungsknoten enthält, der zu einem 2-Kreis gehört.

Eine Kaktus-Repräsentation heißt *cycle-type*, wenn sie keine leere 3-Verzweigung enthält.

Eine *cycle-type normal cactus representation* nennt man auch kurz CNCR.

CNCRs

Lemma (Nagamochi/Kameda)

Angenommen es gibt für eine Teilmenge $\mathcal{C}' \subseteq \mathcal{C}(G)$ der MinCuts eines Graphen G eine Kaktus-Repräsentation $(\mathcal{R}, \varphi)$. Dann gilt:

- *Es gibt eine CNCR für $\mathcal{C}'$.*
- *Die CNCR für $\mathcal{C}'$ ist eindeutig.*
- *Jede CNCR für $\mathcal{C}'$ hat höchstens $|V(G)|$ leere Knoten.*
- *$(\mathcal{R}, \varphi)$ kann in eine CNCR für $\mathcal{C}'$ konvertiert werden in Zeit und Platz linear in der Größe von $(\mathcal{R}, \varphi)$.*

Eigenschaften von Kaktus-Repräsentationen

- Angenommen wir haben zwei Kaktus-Repräsentationen $(\mathcal{R}_1, \varphi_1)$ und $(\mathcal{R}_2, \varphi_2)$ für Teilmengen $\mathcal{C}_1, \mathcal{C}_2$ von $\mathcal{C}(G)$.
- Nagamochi/Kameda haben gezeigt, dass es bei Existenz von Knoten $z_1 \in V(\mathcal{R}_1)$ und $z_2 \in V(\mathcal{R}_2)$ mit

$$\varphi_1^{-1}(z_1) \cup \varphi_2^{-1}(z_2) = V(G)$$

eine Kaktus-Repräsentation $(\mathcal{R}, \varphi)$ für $\mathcal{C}_1 \cup \mathcal{C}_2$ gibt, wo man $\mathcal{R}$ durch Identifikation von Knoten z_1 mit Knoten z_2 (zum neuen Knoten z) erhält und die
Abbildung $\varphi: \; V(G) \to V(\mathcal{R}_1) \cup V(\mathcal{R}_2) \cup \{z\} \setminus \{z_1, z_2\}$ wie folgt definiert ist:

$$\begin{aligned}
\varphi^{-1}(z) &= \varphi_1^{-1}(z_1) \cap \varphi_2^{-1}(z_2) \\
\varphi^{-1}(x_1) &= \varphi_1^{-1}(x_1) \text{ für alle Knoten } x_1 \in V(\mathcal{R}_1) \setminus z_1 \\
\varphi^{-1}(x_2) &= \varphi_2^{-1}(x_2) \text{ für alle Knoten } x_2 \in V(\mathcal{R}_2) \setminus z_2
\end{aligned}$$

Eigenschaften von Kaktus-Repräsentationen

- Die so definierte Repräsentation bezeichnen wir mit $(\mathcal{R}_1, \varphi_1) \oplus (\mathcal{R}_2, \varphi_2) = (\mathcal{R}, \varphi)$ und die entsprechenden Knoten z_1, z_2 heißen verbundene Knoten.
- Der neue Knoten z ist immer ein Artikulationsknoten in $\mathcal{R}$.
  Er ist genau dann leer in $(\mathcal{R}, \varphi)$, wenn $\varphi_1^{-1}(z_1) \cap \varphi_2^{-1}(z_2) = \emptyset$ gilt.

Kritische Kanten und (s,t)-MC-Partition

Eine Kante $e = (s,t)$ heißt *kritisch*, falls $c_G(e) > 0$ und $\lambda_G(s,t) = \lambda_G$.

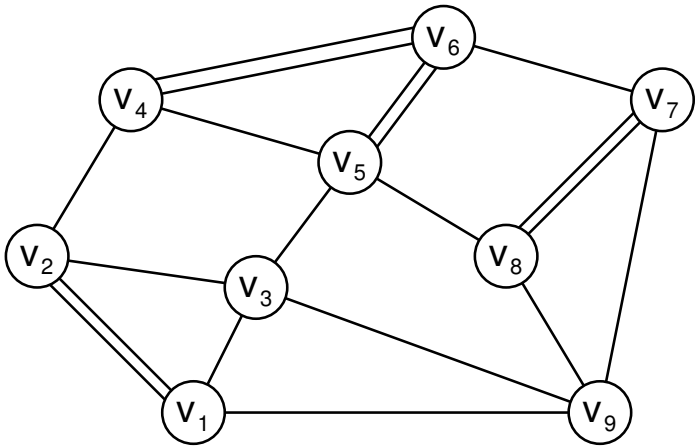
Lemma (Karzanov/Timofeev)

Für jede kritische Kante $e = (s,t)$ sind beliebige MinCuts, die s und t separieren, keine Crossing Cuts.
D.h. es gibt eine geordnete Partition $(V_1, \ldots, V_2)$ von $V(G)$, so dass die Menge der Cuts der Form $\{V_1 \cup V_2 \cup \ldots \cup V_i, V_{i+1} \cup \ldots \cup V_r\}$ gleich der Menge der MinCuts in $\mathcal{C}(G)$ ist, die s und t separieren.

Solch eine geordnete Partition nennt man (s,t) minimum cut o-partition oder kurz (s,t)-MC-Partition.

Beispiel für (s, t)-MC-Partition

Beispiel:



Die (s, t)-MC-Partition für $(s, t) = (v_1, v_9)$ ist $\{V_1 = \{v_1\}, V_2 = \{v_2\}, V_3 = \{v_3\}, V_4 = \{v_4, v_5, v_6\}, V_5 = \{v_7, v_8\}, V_6 = \{v_9\}$.

Berechnung der (s, t)-MC-Partition

Lemma

Sei (s, t) eine kritische Kante in einem Graphen.
Wenn ein beliebiger Maximum Flow zwischen s und t gegeben ist, kann die (s, t)-MC-Partition in Zeit und Platz $\mathcal{O}(m + n)$ berechnet werden.

(Beweis: Karzanov/Timofeev, Naor/Vazirani)

Mit Partition π kompatible und unteilbare Cuts

Sei π eine Partition $\{V_1, V_2, \ldots, V_r\}$ oder eine geordnete Partition $(V_1, V_2, \ldots, V_r)$ von $V(G)$.
Wir nennen einen Cut $\{X, \bar{X}\}$ kompatibel mit π, falls

$$X = \bigcup_{i \in I} V_i \text{ für ein } I \subset \{1, 2, \ldots, r\}$$

Wir nennen einen Cut $\{X, \bar{X}\}$ unteilbar mit π, falls

$$X \subset V_i \text{ für ein } i \in \{1, 2, \ldots, r\}$$

Ein Cut $\{X, \bar{X}\}$ kreuzt eine Partition $\{V_1, V_2, \ldots, V_r\}$ oder eine geordnete Partition $(V_1, V_2, \ldots, V_r)$, falls ein V_i existiert, so dass keine der Teilmengen $X \cap V_i$, $X \setminus V_i$ und $V_i \setminus X$ leer ist.
Jeder Cut, der π nicht kreuzt, ist entweder kompatibel oder unteilbar bezüglich π. Wir bezeichnen die entsprechenden Mengen von MinCuts mit $\mathcal{C}_{\text{comp}}(\pi)$ und $\mathcal{C}_{\text{indv}}(\pi)$

Mit Partition $\pi_{(s,t)}$ kompatible und unteilbare Cuts

Lemma

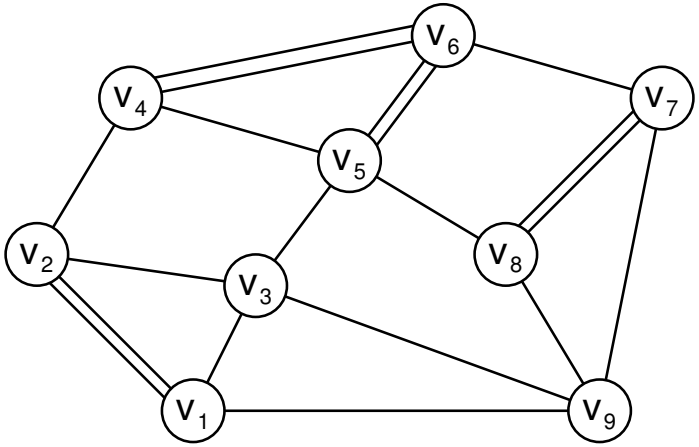
Sei (s,t) eine kritische Kante in einem Graph G und $\pi_{(s,t)}$ die (s,t)-MC-Partition über $\mathcal{C}(G)$. Dann ist jeder MinCut $\{X, \bar{X}\} \in \mathcal{C}(G)$ entweder kompatibel oder unteilbar bezüglich $\pi_{(s,t)}$, d.h. $\mathcal{C}(G) = \mathcal{C}_{comp}(\pi_{(s,t)}) \cup \mathcal{C}_{indv}(\pi_{(s,t)})$.

Man beachte, dass $\mathcal{C}_{\text{comp}}(\pi_{(s,t)})$ einen MinCut enthalten kann, der s und t nicht separiert.

Satz (Nagamochi/Kameda)

Sei (s,t) eine kritische Kante eines Graphen G und $\pi_{(s,t)}$ die (s,t)-MC-Partition.
Dann existiert eine Kaktus-Repräsentation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ für alle MinCuts in $\mathcal{C}_{comp}(\pi_{(s,t)})$, die man (s,t)-Kaktus-Repräsentation nennt.
Weiterhin kann für gegebenes $\pi_{(s,t)}$ die cycle-type normal (s,t)-cactus representation ((s,t)-CNCR) in $\mathcal{O}(m+n)$ Zeit und Platz konstruiert werden.

Mit Partition $\pi_{(s,t)}$ kompatible und unteilbare Cuts



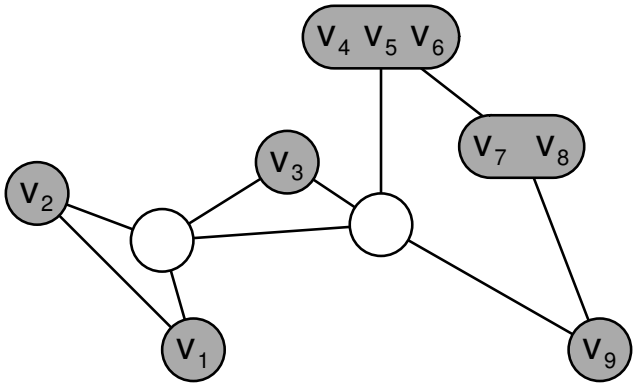


Abbildung: Cycle-type normal (s,t)-cactus representation mit $(s,t) = (v_1, v_9)$ über $\mathcal{C}(G)$

Berechnungsgrundlage

Grundlage für den NNI-Algorithmus:

Satz

In einem Graphen G kann man eine Kante $E = (s, t)$ mit $c_G(e) > 0$, die die folgenden zwei Bedingungen erfüllt, in Zeit $\mathcal{O}(m + n \log n)$ und Platz $\mathcal{O}(m + n)$ berechnen:

1. *$\lambda_G(s, t)$ kann in Zeit $\mathcal{O}(m + n \log n)$ und Platz $\mathcal{O}(m + n)$ berechnet werden.*
2. *Wenn $\lambda_G(s, t) = \lambda_G$, dann kann die (s, t)-CNCR in Zeit $\mathcal{O}(m + n \log n)$ und Platz $\mathcal{O}(m + n)$ berechnet werden.*

Maximum Adjacency Ordering

Definition

Eine totale Ordnung $v_1, v_2, \ldots, v_n$ aller Knoten in $V(G)$ bezeichnet man als Maximum Adjacency Ordering (MAO), falls für alle i, j mit $1 \leq i \leq j \leq n-1$ gilt:

$$w(\{v_1, v_2, \ldots, v_{i-1}\}, v_i) \geq w(\{v_1, v_2, \ldots, v_{i-1}\}, v_j)$$

Verbal: Die Anbindung des Knotens v_i an seine Vorgänger $v_1, v_2, \ldots, v_{i-1}$ ist mindestens so groß wie die Anbindung jedes einzelnen nachfolgenden Knotens $v_{i+1}, \ldots, v_n$ an die Vorgänger von v_i.

Maximum Adjacency Ordering

Lemma

Ein MAO eines Graphen G kann in Zeit $\mathcal{O}(m + n \log n)$ und Platz $\mathcal{O}(m + n)$ berechnet werden.
Für ein MAO $v_1, v_2, \ldots, v_n$ in G gilt für die letzten beiden Knoten v_{n-1}, v_n, dass $\lambda_G(v_{n-1}, v_n) = w(\{v_n\}, V_G \setminus \{v_n\})$.

Beweis des letzten Satzes

Beweis.

- Berechne zuerst ein MAO $v_1, v_2, \ldots, v_n$ von G.
- Wähle den Knoten v_p mit höchstem Index p, so dass v_p und v_n durch eine Kante (mit pos. Gewicht) verbunden sind.
- Sei $s = v_n$ und $t = v_p$.
- Man beachte, dass $v_1, v_2, \ldots, v_p, v_n$ ein MAO in dem Graphen G' ist, der aus G entsteht, wenn man die Knoten $v_{p+1}, \ldots, v_{n-1}$ löscht.
- D.h. nach dem letzten Lemma gilt:
  $\lambda_G(s,t) \geq \lambda_{G'}(s,t) = w_{G'}(\{s\}, V(G') \setminus \{s\}) = w_G(\{s\}, V(G) \setminus \{s\})$
- Da $\lambda_G(s,t) \leq w_G(\{s\}, V(G) \setminus \{s\})$, folgt der 1. Teil des Satzes.

Beweis des letzten Satzes

Beweis.

- Annahme: $\lambda_G(s,t) = \lambda_G$ (wie im 2. Teil des Satzes)
- Ein Maximum Flow zwischen den letzten beiden Knoten eines MAO kann in $\mathcal{O}(m)$ Zeit und Platz berechnet werden (Arikati/Mehlhorn).

$\Rightarrow$ MaxFlow zwischen s und t kann in $\mathcal{O}(m)$ gefunden werden.

- Für kritische Kante (s,t) mit gegebenem MaxFlow zwischen s und t kann nach Lemma die (s,t)-MC-Partition in $\mathcal{O}(m+n)$ Zeit und Platz bestimmt werden.
- Nach vorhergehendem Satz existiert für die kritische Kante (s,t) und die (s,t)-MC-Partition $\pi_{(s,t)}$ eine Kaktus-Repräsentation für alle MinCuts in $\mathcal{C}_{\text{comp}}(\pi_{(s,t)})$, genannt (s,t)-Kaktus-Repräsentation, deren CNCR in $\mathcal{O}(m+n)$ Zeit und Platz konstruiert werden kann.
- Die cycle-type normal (s,t)-cactus representation kann in Zeit $\mathcal{O}(m)$ aus dem MaxFlow berechnet werden.

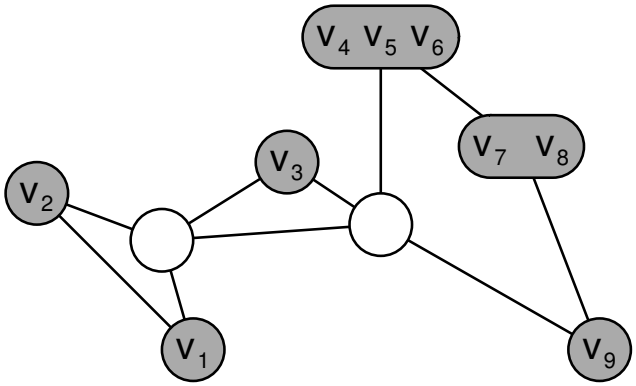
Berechnung der Kaktus-Repräsentation

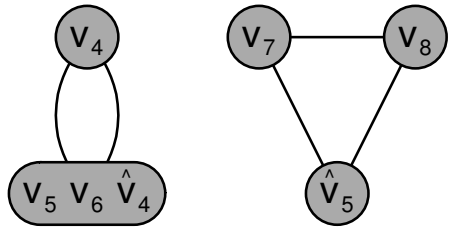
- Sei im folgenden G^* der Eingabe-Graph, für den eine Kaktus-Repräsentation berechnet werden soll.
- Die Konstruktion der Kaktus-Repräsentation von $\mathcal{C}(G^*)$ erfolgt rekursiv auf der Basis des letzten Satzes.
- G/X stellt den Graph dar, den man aus G erhält, wenn man alle Knoten in X zu einem einzigen Knoten kontrahiert.
- Sei $\lambda = \lambda(G^*)$.
- Wir wählen eine Kante (s,t) entsprechend dem letzten Satz.
- Falls $\lambda_G(s,t) > \lambda$, dann kontrahiere Knoten $\{s,t\}$ (kein MinCut in G separiert s und t).
- Falls $\lambda_G(s,t) = \lambda$, dann bestimme (s,t)-MC-Partition $\pi_{(s,t)} = (V_1, \ldots, V_r)$ und (s,t)-Kaktus-Repräsentation $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ in G.
- Nach Lemma sind alle mit $\pi_{(s,t)}$ kompatiblen MinCuts durch $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ repräsentiert und jeder mit $\pi_{(s,t)}$ unteilbare MinCut $\{X, V(G) \setminus X\}$ erfüllt $X \subset V_i$ für ein $V_i \in \pi_{(s,t)}$

Berechnung der Kaktus-Repräsentation

- Sei $G_i = G/(V(G) \setminus V_i)$ der Graph, bei dem die Knoten $V(G) \setminus V_i$ $(i = 1, \ldots, r)$ zu einem Knoten $\hat{v}_i$ kontrahiert wurden, wobei gilt: $\lambda_{G_i} \geq \lambda_G$.
- Annahme: für jedes G_i wurde eine Kaktus-Repräsentation $(\mathcal{R}_i, \varphi_i)$ rekursiv berechnet, wobei $(\mathcal{R}_i, \varphi_i)$ der triviale Kaktus sein soll falls $\lambda_{G_i} > \lambda_G$.
- Dann sind alle MinCuts repräsentiert in $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$, $(\mathcal{R}_1, \varphi_1), \ldots, (\mathcal{R}_r, \varphi_r)$.
- Weiterhin können diese Repräsentationen zu einer einzigen vereinigt werden, indem die Knoten, die $\hat{v}_i$ enthalten, als verbundene Knoten betrachtet werden.

Berechnung der Kaktus-Repräsentation





Kaktus-Repräsentationen $(\mathcal{R}_i, \varphi_i)$, wobei sich in den Fällen $i = \{1, 2, 3, 6\}$ der triviale Kaktus ergibt.
Auf dem Bild sieht man $(\mathcal{R}_4, \varphi_4)$ und $(\mathcal{R}_5, \varphi_5)$.

Berechnung der Kaktus-Repräsentation

- Wenn CACTUS(G', V^{old}) für einen Graph G' während der Ausführung von CACTUS(G'', V^{old}) für einen Graph G'' aufgerufen wird, nennen wir G' ein Kind von G'' und G'' den Vater von G'.
- Die Vater-Kind-Beziehung induziert einen Baum $\mathcal{T}$, der am Eingabe-Graph G^* gewurzelt ist (Aufrufbaum).
- Bemerkung: Falls $w(V_i, V(G) \setminus V_i) = \lambda$, dann bleibt der Cut $\{V_i, \hat{v}_i\}$ ein MinCut in einem Kind G_i, obwohl der Cut schon in seinem Vater G erkannt wurde.
- Derselbe MinCut kann in einem Nachfolger von G_i sein.
- Ein MinCut ist alt in einem Graph G' (also wurde schon in einem Vorfahren entdeckt), nur wenn ein einzelner Knoten v von $V(G') \setminus \{v\}$ separiert wird.
- D.h. wir können testen, ob die (s, t)-Kaktus-Repräsentation neue MinCuts enthält, indem wir die Knoten $v \in V(G')$ als 'alt' markieren, falls v ein kontrahierter Knoten $\hat{v}_i$ im Vorgänger ist.

NNI-Alg. für Kaktus-Repräsentation

Algorithmus 7 : CACTUS(G, V^{old})

Input : Graph G, Teilmenge $V^{\text{old}} \subset V(G)$
Output : Kaktus-Repr. $(\mathcal{R}, \varphi)$ für eine Menge $\mathcal{C}'$ von MinCuts, so dass
$\mathcal{C}(G) \setminus \{\{\bar{v}, V(G) \setminus \{\bar{v}\}\} | \bar{v} \in V^{\text{old}}\} \subseteq \mathcal{C}' \subseteq \mathcal{C}(G)$

if $|V(G)| = 1$ **then return** *Trivial-Kaktus* $(\mathcal{R}, \varphi)$;
else
  Wähle Kante $e = (s, t) \in E(G)$ mit $w_G(e) > 0$;
  if $\lambda_G(s, t) > \lambda$ *oder* (s, t)*-Kaktus Repr.* $(\mathcal{R}_{(s,t)}, \varphi_{(s,t)})$ *repräsentiert keinen anderen Cut außer die folgenden:* $\{\bar{v}, V(G) \setminus \{\bar{v}\}\}, \bar{v} \in V^{old}$ **then**
    $G = G/\{s, t\}$; $V^{\text{old}} = V^{\text{old}} \setminus \{s, t\}$;
    return *CACTUS*(G, V^{old})
  else
    foreach V_i *in der* (s, t)*-MC-Partition* $\pi_{(s,t)} = (V_1, \ldots, V_r)$ **do**
      $G_i = G/(V(G) \setminus V_i)$, wobei $\bar{v}_i$ den Knoten bezeichnet, der durch die Kontraktion von $V(G) \setminus V_i$ entsteht;
      $V_i^{\text{old}} = (V^{\text{old}} \cap V_i) \cup \{\bar{v}_i\}$;
      $(\mathcal{R}_i, \varphi_i) =$CACTUS$(G_i, V_i^{\text{old}})$
    $(\mathcal{R}, \varphi) = (\mathcal{R}_{(s,t)}, \varphi_{(s,t)}) \oplus (\mathcal{R}_1, \varphi_1) \oplus \ldots \oplus (\mathcal{R}_r, \varphi_r)$;
    return $(\mathcal{R}, \varphi)$ *nach Konvertierung in CNCR*

NNI-Alg. für Kaktus-Repräsentation

Algorithmus 8 : CONSTRUCT

Input : Graph G^*
Output : CNCR $(\mathcal{R}, \varphi)$ für $\mathcal{C}(G)$

Compute $\lambda = \lambda(G)$;
$V^{\text{old}} = \emptyset$;
$(\mathcal{R}, \varphi) =$CACTUS(G^*, V^{old})

Satz

Der vorgestellte Algorithmus berechnet eine Kaktus-Repräsentation für alle Minimum Cuts in Zeit $\mathcal{O}(mn + n^2 \log n)$ *und Platz* $\mathcal{O}(m + n)$.

Berechnung eines globalen MinCuts

- 1994 Algorithmus publiziert von M. Stoer und F. Wagner
- benutzt keine MaxFlow-Technik
- sehr einfach
- arbeitet in $n-1$ Phasen
- Phase hat starke Ähnlichkeit zu den Algorithmen von Prim (minimale Spannbäume) bzw. Dijkstra (kürzeste Wege)
- Pro Phase Komplexität $\mathcal{O}(m + n \log n)$
- Gesamtkomplexität $\mathcal{O}(mn + n^2 \log n)$

Der Stoer/Wagner-Algorithmus

Algorithmus 9 : MinCut Berechnung nach Stoer & Wagner

Input : Ungerichteter Graph $G = (V, E)$
Output : Ein MinCut $C_{\min}$ entsprechend $\lambda(G)$

```
Wähle einen beliebigen Startknoten a;
C_min ← undefiniert;
V' ← V;
while |V'| > 1 do
    A ← {a};
    while A ≠ V' do
        Füge zu A den am meisten angebundenen Knoten hinzu;
        Aktualisiere die Kapazitäten zwischen A und den Knoten in V' \ A;
    C := Schnitt von V', der den zuletzt zu A hinzugefügten Knoten vom Rest des Graphen
    trennt;
    if C_min = undefiniert or w(C) < w(C_min) then
        C_min ← C;
    Verschmelze die zwei Knoten, die zuletzt zu A hinzugefügt wurden;
return C_min;
```

Beschreibung des Stoer/Wagner-Algorithmus

- Wahl eines beliebigen Startknotens a
- Algorithmus verwaltet eine Knotenteilmenge A (initialisiert mit a), die immer wieder um einen Knoten erweitert wird, der gerade maximale Anbindung (also Summe der Kantenkapazitäten) an die aktuellen Knoten in A hat.
- Nachdem alle Knoten in A eingefügt wurden, nennt man den Cut, der nur den zuletzt hinzugefügten Knoten t vom Rest abtrennt 'Cut der Phase'.
- Verschmelzung der beiden zuletzt behandelten Knoten s und t:
  - Wenn zwischen s und t eine Kante existiert, wird sie einfach gelöscht.
  - Alte Kanten von einem anderen Knoten $\{x, s\}$ und $\{x, t\}$ werden durch eine neue Kante mit der Summe der alten Gewichte $w(s, x) + w(t, x)$ ersetzt.

Korrektheit des Stoer/Wagner-Algorithmus

Lemma

Der 'Cut der Phase' ist ein Minimum (s,t)-Cut für die beiden zuletzt in A eingefügten Knoten s und t.

Beweis.

- Betrachte beliebigen s-t-Cut C für die letzten zwei Knoten.
- Ein Knoten $v \neq a$ heißt aktiv, falls sich v und sein unmittelbarer Vorgänger bezüglich der Addition zu A in unterschiedlichen Teilen von C befinden.
- Sei A_v die Menge von Knoten, die in A sind bevor v hinzugefügt wird.
- Sei $w(S,v)$ für eine Knoten(teil)menge S die Kapazitätssumme aller Kanten zwischen v und den Knoten in S.

Korrektheit des Stoer/Wagner-Algorithmus

Beweis.

- Beweisidee: (Induktion über die aktiven Knoten)

  Für jeden aktiven Knoten v gilt, dass die Anbindung (Adjazenz) zu den Knoten, die zuvor eingefügt wurden (also A_v), nicht das Gewicht des durch C in $A_v \cup \{v\}$ induzierten Cuts (bezeichnet mit C_v) überschreitet.

- Es gilt also zu zeigen, dass gilt:

$$w(A_v, v) \leq w(C_v)$$

- Induktionsanfang: (1. aktiver Knoten)
  Im Basisfall ist die Ungleichung erfüllt, weil beide Werte für den ersten aktiven Knoten gleich sind.

Korrektheit des Stoer/Wagner-Algorithmus

Beweis.

- Induktionsvoraussetzung:
  Das Lemma stimmt für alle aktiven Knoten bis zum aktiven Knoten v.

$\Rightarrow$ Der Wert für den nächsten aktiven Knoten u ist

$$\begin{aligned} w(A_u, u) &= w(A_v, u) + w(A_u \setminus A_v, u) \\ &\leq w(A_v, v) + w(A_u \setminus A_v, u) \\ &\leq w(C_v) + w(A_u \setminus A_v, u) \qquad \text{(Ind.voraussetzung)} \\ &\leq w(C_u) \end{aligned}$$

- Die letzte Zeile folgt, weil alle Kanten zwischen $A_u \setminus A_v$ und u ihr Gewicht zu $w(C_u)$ beitragen, aber nicht zu $w(C_v)$.
- Da t durch den Cut C von seinem unmittelbaren Vorgänger s getrennt wird, ist es immer ein aktiver Knoten.
- Die Schlussfolgerung $w(A_t, t) \leq w(C_t)$ beschließt den Beweis.

□

Korrektheit des Stoer/Wagner-Algorithmus

Satz

Ein 'Cut der Phase' mit minimaler Kantenkapazität ist ein MinCut des gegebenen Graphen

Beweis.

- Für $|V| = 2$ trivial (bei zwei Knoten existiert nur *ein* Cut).
- Annahme: $|V| > 2$, 2 Fälle:
  1. Entweder der Graph hat einen MinCut, der gleichzeitig ein (lokaler) Minimum s-t-Cut ist.
     Dann ist der entsprechende Cut nach Lemma ein globaler MinCut des Graphen.
  2. Andernfalls hat der Graph einen globalen MinCut, bei dem s und t nicht separiert werden (sich also auf der gleichen Seite befinden).
     Dann wird der globale MinCut durch das Verschmelzen von s und t nicht verändert.

$\Rightarrow$ (Induktion über die Anzahl der Knoten)
Der 'Cut der Phase' mit minimaler Gewichtssumme ist ein globaler MinCut.

□

Beispiel für den Stoer/Wagner-Algorithmus

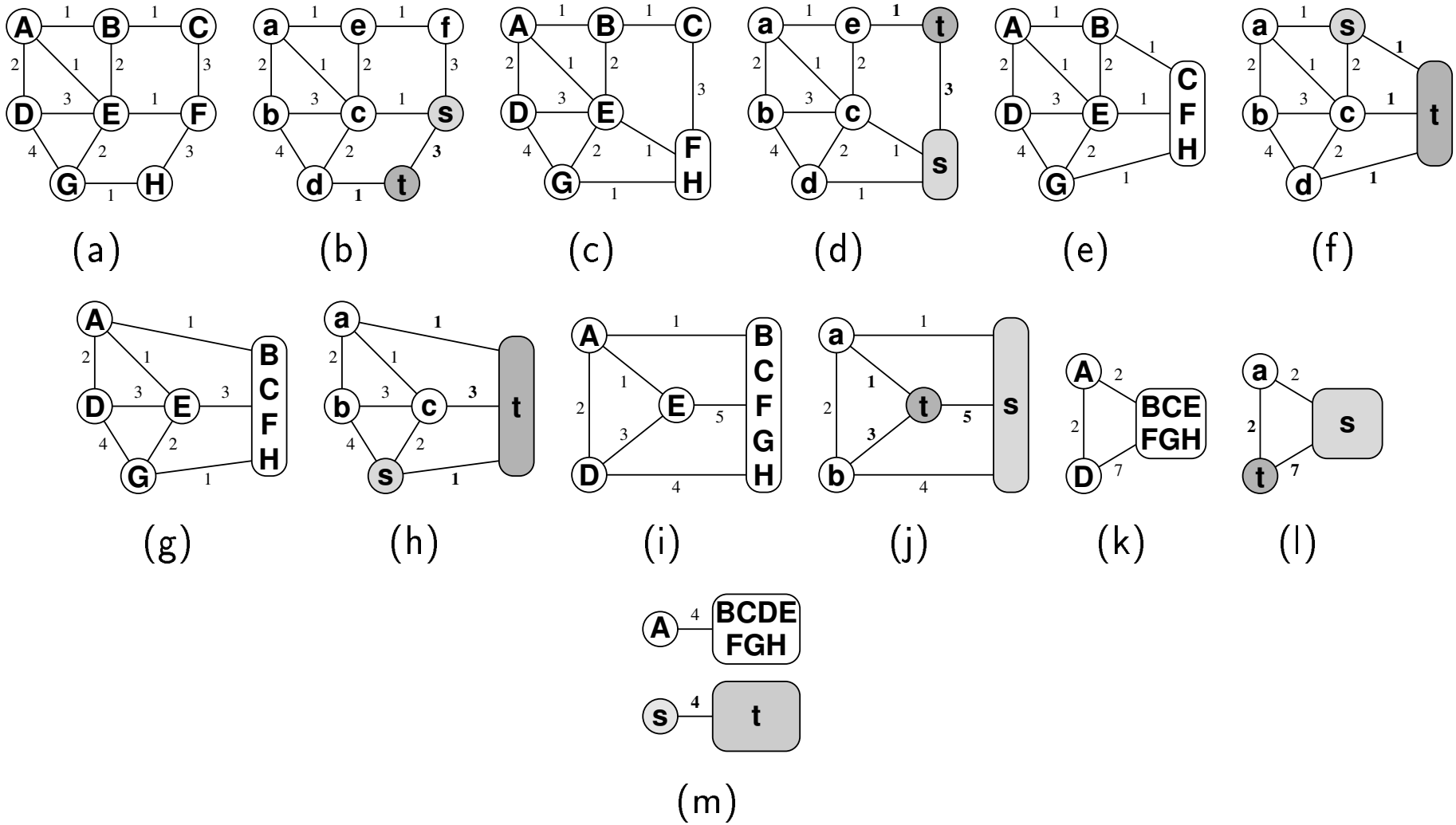


Abbildung: Der MinCut $\{ABDEG\} \mid \{CFH\}$ wird gefunden in Phase 3 (f)

Der randomisierte MinCut-Algorithmus von Karger

- Gesucht: globaler Minimum Cut in einem Multi-Graphen
- D. Karger (1992): Vorschlag eines sehr einfachen Algorithmus ohne augmentierende Pfade und Flussberechnungen
- Ansatz: randomisierte Suche (Monte-Carlo-Algorithmus), die mit gewisser (Erfolgs-)Wahrscheinlichkeit den Minimum Cut liefert (und mit gewisser (Fehler-)Wahrscheinlichkeit einen beliebigen anderen Cut)

Der randomisierte MinCut-Algorithmus von Karger

- Der Algorithmus wählt eine beliebige Kante $e = \{u, v\}$ von G, und zwar uniform identisch verteilt (d.h. jede Kante wird mit gleicher Wahrscheinlichkeit gezogen).
- Die gewählte Kante wird kontrahiert, d.h. es wird ein Multi-Graph G' erzeugt, bei dem die Knoten u und v zu einem neuen Knoten w verschmolzen sind.
- Kanten zwischen u und v verschwinden.
- Andere Kanten bleiben erhalten. Wenn genau ein Endknoten entweder u oder v ist, dann endet dafür eine neue Kante am (Super-)Knoten w.
- Hinweis: auch wenn G keine Multikanten enthält, kann G' welche enthalten.
- Das Verfahren wird rekursiv auf G' angewendet.

Der randomisierte MinCut-Algorithmus von Karger

- Der Algorithmus endet, wenn nur noch zwei (Super-)Knoten vorhanden sind.
- Jeder der beiden Knoten enthält eine gewisse Menge der Knoten des ursprünglichen Graphen.
- Diese Partition definiert einen Cut, der vom Algorithmus ausgegeben wird.

Satz

Der Monte-Carlo-Algorithmus von Karger gibt mit Wahrscheinlichkeit $\geq 1/\binom{n}{2}$ *den globalen Minimum Cut des Multigraphen zurück.*

- Nachdem es exponentiell viele Cuts in G gibt, würde man vermuten, dass die Wahrscheinlichkeit, den globalen MinCut zu finden, exponentiell klein ist.
- Was favorisiert die kleinen Cuts?

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Betrachte globalen MinCut (A, B) in G.
- Sei die Schnitt-Kardinalität k, d.h. es gibt genau k Kanten (F), die einen Endpunkt in A und einen in B haben.
- gesucht: untere Schranke für die Wahrscheinlichkeit, dass der Algorithmus (A, B) zurück gibt.
- Was kann dabei schiefgehen?
- Eine Kante aus F könnte kontrahiert werden.
- Dann würden zwei Knoten kontrahiert, die auf unterschiedlichen Seiten des Cuts (A, B) liegen.
- Der Algorithmus könnte nicht mehr (A, B) ausgeben.
- Wenn eine beliebige Kante aus $E \setminus F$ kontrahiert wird, besteht noch die Chance auf die Ausgabe (A, B).

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- also gesucht: obere Schranke für die Wahrscheinlichkeit, dass eine Kante aus F kontrahiert wird.
- Wir brauchen eine untere Schranke für die Kardinalität von E.
- Wenn der globale MinCut den Wert $\lambda(G) = k$ hat, gilt für den Minimalgrad des Graphen $\delta(G) \geq \lambda(G) = k$.
- Es gilt also $|E| \geq kn/2$.

$\Rightarrow$ Die Wahrscheinlichkeit, dass eine Kante aus F kontrahiert wird, ist

$$\frac{k}{kn/2} = \frac{2}{n}$$

- Betrachte jetzt die Situation nach j Iterationen.
- Es gibt $n - j$ Superknoten im aktuellen G'.

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Annahme: es wurde noch keine Kante aus F kontrahiert.
- Da jeder Cut in G' auch ein Cut in G ist, sind zu jedem Superknoten in G' mindestens k Kanten inzident.
- D.h. G' hat mindestens $k(n-j)/2$ Kanten.

$\Rightarrow$ Die Wahrscheinlichkeit, dass eine Kante aus F in der nächsten Iteration $j+1$ kontrahiert wird, ist

$$\frac{k}{k(n-j)/2} = \frac{2}{n-j}$$

- Cut (A, B) wird ausgegeben, wenn in Iteration $1, \ldots, n-2$ keine Kante aus F kontrahiert wird.

Randomisierter MinCut-Algorithmus: Analyse

Beweis.

- Sei $\mathcal{E}_j$ das Ereignis, dass in Iteration j keine Kante aus F kontrahiert wird. Dann gilt also: $\Pr[\mathcal{E}_1] \geq 1 - 2/n$ und

$$\Pr[\mathcal{E}_{j+1} | \mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_j] \geq 1 - 2/(n-j)$$

- gesucht: unter Schranke für $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_{n-2}]$ bzw. $\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_{n-2} | \mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_{n-3}]$

$$\begin{aligned} &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \ldots \left(1 - \frac{2}{n-j}\right) \ldots \left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{2 \cdot 1}{n(n-1)} = \binom{n}{2}^{-1} \end{aligned}$$

Randomisierter MinCut-Algorithmus: Analyse

- Wahrscheinlichkeit, dass der randomisierte MinCut-Algorithmus nicht den Cut (A, B) ausgibt, ist höchstens $1 - 1/\binom{n}{2}$
- Nach $\binom{n}{2}$ Läufen mit unabhängigen Zufallsentscheidungen ist die Wahrscheinlichkeit, dass nie der MinCut gefunden wurde, höchstens

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leq \frac{1}{e}$$

- Wenn $c\binom{n}{2} \log n$ Läufe gestartet werden, sinkt die Fehlerwahrscheinlichkeit auf $\leq e^{-c \log n} = 1/n^c$.
- Laufzeit ist in dieser einfachen Form noch relativ hoch, verglichen mit dem besten deterministischen Algorithmus

Randomisierter MinCut-Algorithmus: Analyse

- Ansatz ist unbalanciert: Fehlerwahrscheinlichkeit erhöht sich zum Ende eines Laufs
- Wenn man den Algorithmus nur t Schritte laufen lässt, ist die Wahrscheinlichkeit, dass keine Kante aus F kontrahiert wird $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_t]$ bzw. $\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2|\mathcal{E}_1] \cdot \ldots \cdot \Pr[\mathcal{E}_t|\mathcal{E}_1 \cap \mathcal{E}_2 \cap \ldots \cap \mathcal{E}_{t-1}]$

$$
\begin{aligned}
&\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\ldots\left(1-\frac{2}{n-(t-1)}\right)\\
&= \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdot\ldots\cdot\frac{n-t}{n-t+2}\cdot\frac{n-t-1}{n-t+1}\\
&= \frac{(n-t)\cdot(n-t-1)}{n(n-1)} \in \Omega\left(\frac{(n-t)^2}{n^2}\right)
\end{aligned}
$$

Randomisierter MinCut-Algorithmus: Analyse

Verbesserte Variante:

- Lasse die einfache Variante mehrmals laufen
  (z.B. 2-mal bis $\lceil n/\sqrt{2}+1 \rceil$ Knoten
  oder 4-mal bis $n/2$ Knoten)
- Setze jeden der Läufe rekursiv mit der modifizierten Variante fort.
- Laufzeit: $\mathcal{O}(n^2 \log n)$ mit Erfolgswahrscheinlichkeit $\Omega(1/\log n)$
- $\mathcal{O}(\log n)$-malige Wiederholung des modifizierten Algorithmus sorgt für konstante Erfolgswahrscheinlichkeit.
- Um eine kleine Fehlerwahrscheinlichkeit ϵ zu erreichen, kann man $\mathcal{O}(\log n \log \epsilon^{-1})$ Wiederholungen durchführen, was zu einer Gesamtlaufzeit von $\mathcal{O}(n^2 \log^2 n \log \epsilon^{-1})$ führt.

All-Pairs MaxFlow / MinCut

- gegeben: Graph $G = (V, E)$ mit n Knoten, p ausgewählte Knoten
- gesucht: (lokale) MaxFlow/MinCut-Werte für alle Paare aus den p gegebenen Knoten

$\Rightarrow$ kann einfach durch Lösen von $\binom{p}{2} = \frac{1}{2}p(p-1)$ MaxFlow-Problemen berechnet werden.

- Kann in ungerichteten Graphen aber auch mit nur $p - 1$ MaxFlow-Berechnungen gelöst werden.

Equivalent Flow Trees

Sei G also ungerichtet und sei $v_{ij} = v_{ji}$ der Wert eines MaxFlows zwischen den Knoten i und j.

Lemma

1. *Für alle Knoten* $i, j, k \in \{1, \ldots, n\}$ *gilt:*

$$v_{ik} \geq \min\{v_{ij}, v_{jk}\}$$

2. *Es gibt einen Baum* T *auf den Knoten* 1 *bis* n*, so dass für alle Paare von Knoten* i, j *gilt:*

$$v_{ij} = \min\{v_{ij_1}, v_{j_1 j_2}, \ldots, v_{j_k j}\},$$

wobei $i - j_1 - \cdots - j_k - j$ *der (eindeutige) Pfad von Knoten* i *zu Knoten* j *in* T *ist.*

Equivalent Flow Trees

Beweis.

1. Sei $(X, \bar{X})$ ein i,k-MinCut, d.h. $v_{ik} = w(X, \bar{X})$.
   - Falls $j \in \bar{X}$, dann gilt $v_{ij} \leq w(X, \bar{X}) = v_{ik}$.
   - Falls $j \in X$, dann gilt $v_{jk} \leq w(X, \bar{X}) = v_{ik}$.
2. Betrachte K_n (den vollständigen Graphen auf n Knoten), bei dem jede Kante (i,j) mit v_{ij} gewichtet ist und sei T darin ein Spannbaum maximalen Gewichts.
   - Induktiv folgt aus dem vorangegangenen Punkt, dass für jedes Knotenpaar (i,j) gilt: $v_{ij} \geq \min\{v_{ij_1}, v_{j_1 j_2}, \ldots, v_{j_k j}\}$, wobei $i - j_1 - \cdots - j_k - j$ der (eindeutige) Pfad von Knoten i zu Knoten j in T ist.
   - Angenommen, für ein Paar (i,j) gilt echte Ungleichheit. Dann gibt es eine Kante (i',j') auf dem Pfad zwischen i und j mit $v_{ij} > v_{i'j'}$. Das würde bedeuten, dass der Baum T' der aus T durch Tausch von (i',j') gegen (i,j) entsteht, ein größeres Gewicht hätte. (Widerspruch)

□

Algorithmus von Gomory und Hu

- Die Existenz eines Equivalent Flow Trees hat natürlich gleichzeitig die Konsequenz, dass unter den $\binom{n}{2}$ MaxFlow- bzw. MinCut-Werten v_{ij} nur höchstens $n-1$ verschiedene Werte existieren können (nämlich die Kantengewichte von T).
- Außerdem kommt man so zu der Vermutung, dass $n-1$ MaxFlow-Berechnungen genügen, um einen solchen Equivalent Flow Tree T zu konstruieren und damit alle v_{ij}-Werte zu berechnen. Ein Algorithmus von Gomory und Hu erreicht dies tatsächlich.

Algorithmus von Gomory und Hu

- Gegeben zwei Knoten i, j und ein i,j-MinCut $(X, \bar{X})$.
- Definiere den kontrahierten Graph G^c als den Graph, den man aus G erhält, indem man die Knoten in $\bar{X}$ zu einem einzelnen (speziellen) Knoten $u_{\bar{X}}$ zusammenfasst und für jeden Knoten $v \in X$ alle über den Schnitt führenden Kanten durch eine einzelne Kante $\{v, u_{\bar{X}}\}$ mit der entsprechend summierten Gesamtkapazität ersetzt.

Lemma

Für jedes Paar von (normalen) Knoten i', j' in G^c (d.h. $i', j' \in X$) hat der MaxFlow bzw. MinCut zwischen i' und j' in G^c den gleichen Wert wie der MaxFlow/MinCut im Original-Graphen.
(Entsprechendes gilt natürlich für Knoten $i', j' \in \bar{X}$, wenn man X in G kontrahiert.)

Algorithmus von Gomory und Hu

Corollary

Für jedes Paar von Knoten $i', j' \in X$ (oder $\bar{X}$) gibt es einen Minimum i', j'-Cut, so dass sich alle Knoten von $\bar{X}$ (bzw. X) auf der gleichen Seite des Schnitts befinden.

- Für zwei Knoten i, j und einen i, j-MinCut $(X, \bar{X})$ repräsentiert man die aktuelle Situation durch einen Baum mit zwei Knoten (die für X und $\bar{X}$ stehen) und einer Kante mit dem Gewicht v_{ij}. Die Kante repräsentiert dabei den Schnitt $(X, \bar{X})$.
- Sei A die Menge der p zu betrachtenden Knoten, für die die MaxFlow/MinCut-Werte berechnet werden sollen..
- Die ersten beiden Knoten i und j werden aus A gewählt.

Algorithmus von Gomory und Hu

Definition

Ein Baum T wird als Semi-Cut Tree bezeichnet, falls er folgende Eigenschaften besitzt:

1. Jeder Knoten U von T entspricht einer Teilmenge der Knoten von G und enthält mindestens einen Knoten der Menge A.
2. Jede Kante (U, V) ist mit einem Label v versehen, so dass es Knoten $i, j \in A$ mit $i \in U$ und $j \in V$ gibt und der MaxFlow/MinCut zwischen i und j den Wert v hat.
3. Jede Kante (U, V) repräsentiert einen i, j-MinCut mit $i, j \in A$ und i ist in einem Knoten des Teilbaums auf der Seite von U enthalten und j ist in einem Knoten des Teilbaums auf der Seite von V enthalten (und die zwei Teile des Cut bestehen aus der jeweiligen Knotenmenge).

Wenn jeder Knoten eines Semi-Cut Trees T genau einen Knoten der Menge A enthält, dann bezeichnet man T als Cut Tree für A.

Algorithmus von Gomory und Hu

Satz

Sei T ein Cut Tree für A.
Dann gilt für jedes Paar $i, j \in A$: $v_{ij} = \min\{v_1, \ldots, v_{k+1}\}$, wobei v_1 bis v_{k+1} die Kantengewichte in T auf dem Pfad vom Knoten, der i enthält, zum Knoten, der j enthält, sind.

Beweis.

- Sei $i - j_1 - \ldots - j_k - j$ die Folge von A-Knoten, die dem Pfad in T vom Knoten mit i zum Knoten mit j entsprechen.
- Aufgrund der Eigenschaften des Cut Trees sind die Kantenlabels einfach $v_{ij_1}, \ldots, v_{j_k j}$.
- Aufgrund des Lemmas gilt: $v_{ij} \geq \min\{v_{ij_1}, \ldots, v_{j_k j}\}$.
- Andererseits entspricht jede Kante einem i, j-Schnitt, wobei die Kapazität dem Kantenlabel entspricht. Es gilt also: $v_{ij} \leq \min\{v_{ij_1}, \ldots, v_{j_k j}\}$ und damit $v_{ij} = \min\{v_{ij_1}, \ldots, v_{j_k j}\}$.

Algorithmus von Gomory und Hu

Satz

Ein Cut Tree für A existiert und kann mit Hilfe von nur $p-1$ MaxFlow-Brechnungen konstruiert werden.

Beweis.

- Angenommen, wir haben einen Semi-Cut Tree T für A und T hat noch einen Knoten U, der zwei Knoten $i, j \in A$ enthält.
- Aufgrund des Lemmas hat der MaxFlow zwischen i und j in G genau den gleichen Wert wie der MaxFlow zwischen i und j in dem (kontrahierten) Graph G^c, der entsteht, wenn man die Knotenmengen in jedem zu U verbundenen Teilbaum zu einem einzelnen (speziellen) Knoten kontrahiert und für jeden (Original-)Knoten $v \in U$ die Kanten, die in die jeweiligen Teilbäume führen, durch einzelne Kanten zu den entsprechenden neuen (speziellen) Knoten mit aufsummiertem Kantengewicht ersetzt.

Algorithmus von Gomory und Hu

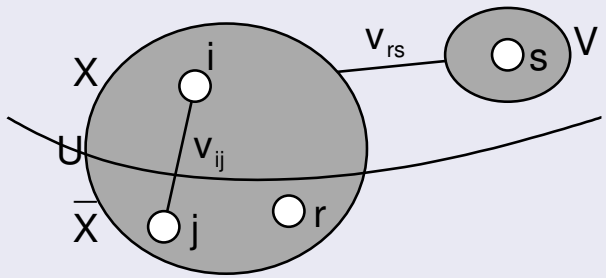
Beweis.

- Sei $(X, \bar{X})$ ein i,j-MinCut in G^c (notwendigerweise mit Kapazität v_{ij}).
- Konstruiere einen Baum T' wie folgt:
  - Spalte U in zwei Knoten X und $\bar{X}$.
  - Verbinde die Knoten X und $\bar{X}$ durch eine Kante mit Label v_{ij} und verbinde jeden Nachbarn V von U entweder zu X oder zu $\bar{X}$, in Abhängigkeit von der Seite des Cuts, auf der sich der zu V's Teilbaum gehörende spezielle Knoten in G^c befunden hat (mit Original-Kantenlabel).
- Die Behauptung ist nun, dass T' auch wieder ein Semi-Cut Tree für A ist.

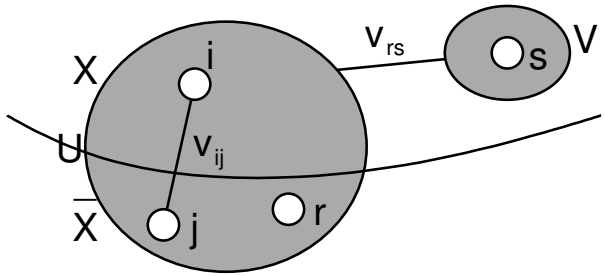
Algorithmus von Gomory und Hu

Beweis.

- Die einzige nicht-triviale zu überprüfende Eigenschaft ist der zweite Teil der Definition.
- Sei also V irgendein Nachbar von U in T und entspreche das Label der Kante (U, V) einem MaxFlow/MinCut-Wert zwischen $r \in U$ und $s \in V$ $(r, s \in A)$.
- Sei o.B.d.A. $j, r \in \bar{X}$, dann können folgende zwei Fälle auftreten:
  1. V wird mit $\bar{X}$ verbunden.
     Die Eigenschaften des Semi-Cut Trees bleiben erhalten.
  2. V wird mit X verbunden.



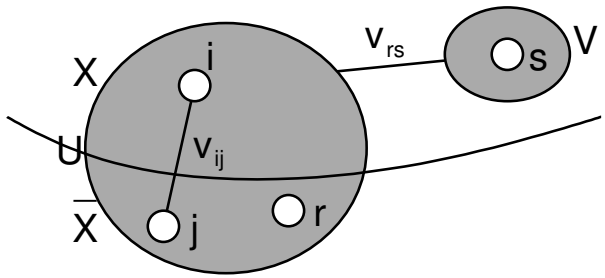
Algorithmus von Gomory und Hu



Beweis.

- Das Label v_{ij} für die Kante $\{X, \bar{X})$ entspricht der Definition.
  Betrachte nun das Label (v_{rs}) der Kante (X, V).
  Es wird gezeigt, dass $v_{is} = v_{rs}$:
  - $v_{is} \geq \min\{v_{ij}, v_{jr}, v_{rs}\}$ (Lemma).
  - Da i und s auf der gleichen Seite des Cuts sind, können die MaxFlow/MinCut-Werte zwischen Knoten in $\bar{X}$ nicht den Wert von v_{is} beeinflussen (v_{jr} ist also egal) und es gilt $v_{is} \geq \{v_{ij}, v_{rs}\}$.

Algorithmus von Gomory und Hu



Beweis.

- Andererseits muss v_{is} durch den zur Kante (U, V) in T gehörigen MinCut beschränkt sein, d.h. $v_{is} \leq v_{rs}$.
  Aufgrund des Lemmas gilt $v_{ij} \geq \min\{v_{is}, v_{rs}\}$, also $v_{ij} \geq v_{rs}$.
  Daraus folgt $v_{is} = v_{rs}$ und die Kante (X, V) entspricht dem MaxFlow zwischen i und s (mit Wert und Schnitt-Mengen wie im Original-Graph).

- Eine wiederholte Aufspaltung liefert den Cut Tree mit Hilfe von $p - 1$ MaxFlow-Berechnungen.

Unit Capacity Networks

Definition

- Ein Graph wird als Unit Capacity Network (oder 0-1 *Network*) bezeichnet, falls die Kapazität aller Kanten gleich 1 ist.
- Ein Unit Capacity Network ist vom Typ 1, falls es keine parallelen Kanten hat.
- Es ist vom Typ 2, falls für jeden Knoten v ($v \neq s$, $v \neq t$) entweder der Eingangsgrad $d^-(v)$ oder der Ausgangsgrad $d^+(v)$ gleich 1 ist.

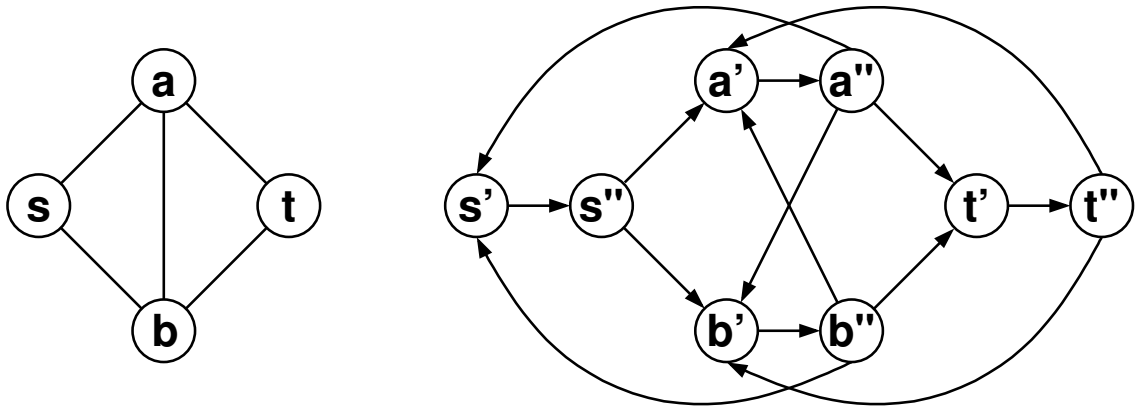
Unit Capacity Networks

Lemma

- *Ein MaxFlow/MinCut kann für ein Unit Capacity Network (mit Dinitz' Algorithmus) in Zeit $\mathcal{O}(m^{3/2})$ berechnet werden.*
- *Für Unit Capacity Networks vom Typ 1 ist die Zeitkomplexität von Dinitz' Algorithmus $\mathcal{O}(n^{2/3}m)$.*
- *Für Unit Capacity Networks vom Typ 2 ist die Zeitkomplexität von Dinitz' Algorithmus $\mathcal{O}(n^{1/2}m)$.*

(Beweis: siehe Shimon Even, Graph Algorithms, 1979)

Ungerichtete ungewichtete Graphen



- Gegeben: ungerichteter (ungewichteter) Graph $G = (V, E)$ mit n Knoten und m Kanten
- Konstruiere gerichteten Graph $\bar{G} = (\bar{V}, \bar{E})$ mit $|\bar{V}| = 2n$ und $|\bar{E}| = 2m + n$ wie folgt:
  - Ersetze jeden Knoten $v \in V$ durch zwei Knoten $v', v'' \in \bar{V}$, verbunden durch eine (interne) Kante $e_v = (v', v'') \in \bar{E}$.
  - Ersetze jede Kante $e = (u, v) \in E$ durch zwei (externe) Kanten $e' = (u'', v')$ und $e'' = (v'', u') \in \bar{E}$.

Ungerichtete ungewichtete Graphen

- $\kappa(s,t)$ wird nun berechnet als MaxFlow in $\bar{G}$ von Quelle s'' zu Senke t' mit Unit Capacity-Kanten
- Hinweis: $c(e_v) = 1$, $c(e') = c(e'') = \infty$ führt übrigens zum gleichen Ergebnis.
- Für jedes Paar $v', v'' \in \bar{V}$, das einen internen Knoten $v \in V$ repräsentiert, ist die interne Kante (v', v'') die einzige von v' ausgehende Kante und die einzige eingehende Kante von v''. Der Graph $\bar{G}$ ist also ein UCN vom Typ 2.
- Nach dem Lemma kann die Berechnung des MaxFlow bzw. des lokalen Knotenzusammenhangs in Zeit $\mathcal{O}(\sqrt{n}m)$ erfolgen.

Ungerichtete ungewichtete Graphen

Trivialer Algorithmus zur Bestimmung von $\kappa(G)$:

- Bestimme Minimum aller lokalen Knotenzusammenhangszahlen.
- Für die Endknoten jeder Kante (s, t) in G gilt:

$$\kappa_G(s, t) = n - 1$$

- Anzahl notwendiger MaxFlow-Berechnungen:

$$\frac{n(n-1)}{2} - m$$

Ungerichtete ungewichtete Graphen

Besserer Algorithmus zur Bestimmung von $\kappa(G)$:

- Betrachte minimalen Knoten-Separator $S \subset V$, der eine 'linke' Knotenteilmenge $L \subset V$ von einer 'rechten' Teilmenge $R \subset V$ separiert.
- Man könnte $\kappa(G)$ berechnen, indem man einen Knoten s in einer Teilmenge (L oder R) fixiert und die lokalen Zusammenhangszahlen $\kappa_G(s,t)$ für alle Knoten $t \in V \setminus \{s\}$ berechnet, wobei einer dieser Knoten auf der anderen Seite des Schnitts liegen muss.
- Problem: wie wählt man einen Knoten s, so dass s nicht zu jedem Minimum Vertex Separator gehört?
- Da $\kappa(G) \leq \delta(G)$, könnte man $\delta(G)+1$ Knoten für s versuchen. Einer davon kann nicht Teil aller Minimum Knoten-Separatoren sein.
- Der Algorithmus hat Komplexität $\mathcal{O}((\delta+1)\cdot(n-1)\cdot\sqrt{nm})) = \mathcal{O}(\delta n^{3/2} m)$

Knotenzusammenhang (Even & Tarjan)

Algorithmus 10 : Knotenzusammenhang κ (Even & Tarjan)

Input : Ungerichteter Graph $G = (V, E)$
Output : $\kappa(G)$

$\kappa_{\min} \leftarrow n - 1$;
$i \leftarrow 1$;
while $i \leq \kappa_{\min}$ **do**
  for $j \leftarrow i + 1$ **to** n **do**
    if $i > \kappa_{\min}$ **then**
      break;
    else if $\{v_i, v_j\} \notin E$ **then**
      Berechne $\kappa_G(v_i, v_j)$ mit MaxFlow-Prozedur;
      $\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v_i, v_j)\}$;
  $i \leftarrow i + 1$;
return $\kappa_{\min}$;

Knotenzusammenhang (Even & Tarjan)

Even/Tarjan-Algorithmus zur Berechnung des (globalen) Knotenzusammenhangs κ

- stoppt die Berechnung der lokalen Knotenzusammenhangszahlen $\kappa_G(v_i, v_j)$, falls das Minimum unter die Anzahl der momentan betrachteten Knoten i fällt
- Algorithmus betrachtet höchstens $\kappa + 1$ Knoten in der Schleife für Variable i
- Jeder Knoten hat mindestens $\delta(G)$ Nachbarn, also höchstens $n - \delta - 1$ Nicht-Nachbarn.

$\Rightarrow$ Es gibt maximal $\mathcal{O}((n - \delta - 1)(\kappa + 1))$ Aufrufe für die Berechnung des lokalen Zusammenhangs (MaxFlow für zwei gegebene Knoten).

$\Rightarrow$ Da $\kappa \leq \delta \leq \bar{d} = 2m/n$ wird der richtige Wert spätestens in Aufruf $2m/n + 1$ gefunden. Die Komplexität ist also $\mathcal{O}(\sqrt{n}m^2)$.

Knotenzusammenhang (Esfahanian & Hakimi)

Verbesserung von Esfahanian & Hakimi:

Lemma

Wenn ein Knoten v zu allen Knoten-Separatoren minimaler Kardinalität gehört, dann gibt es für jeden Minimum Vertex-Cut S zwei Knoten $l \in L_S$ und $r \in R_S$, die zu v adjazent sind.

Knotenzusammenhang (Esfahanian & Hakimi)

Beweis.

- Annahme: v ist an allen Minimum Vertex-Cutsets beteiligt.
- Betrachte die beiden (getrennten) Teile L und R des Restgraphen, der nach dem Löschen verbleibt.
- Jede der beiden Seiten muss einen Nachbarn von v enthalten, sonst wäre v nicht nötig, um die Teile zu trennen
  (und die Knotenmenge wäre damit kein minimaler Separator)
- Jede Seite, die mehr als einen Knoten enthält, muss sogar zwei Nachbarn von v enthalten, da man sonst durch Ersetzen von v durch den einzigen Nachbarn einen MinCut ohne v konstruieren könnte (Widerspruch zur Annahme).

□

Knotenzusammenhang (Esfahanian & Hakimi)

Algorithmus 11 : Knotenzusammenhang κ (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$
Output : $\kappa(G)$

$\kappa_{\min} \leftarrow n - 1$;
Wähle $v \in V$ mit minimalem Grad, also $d(v) = \delta(G)$;
Seien die Nachbarn $N(v) = \{v_1, v_2, \ldots, v_\delta\}$;

foreach *Nicht-Nachbar* $w \in V \setminus (N(v) \cup \{v\})$ **do**
    Berechne $\kappa_G(v, w)$ mit MaxFlow-Prozedur;
    $\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v, w)\}$;

$i \leftarrow 1$;
while $i \leq \kappa_{\min}$ **do**
    for $j \leftarrow i + 1$ **to** $\delta - 1$ **do**
        if $i \geq \delta - 2$ *or* $i > \kappa_{\min}$ **then**
            return $\kappa_{\min}$;
        else if $\{v_i, v_j\} \notin E$ **then**
            Berechne $\kappa_G(v_i, v_j)$ mit MaxFlow-Prozedur;
            $\kappa_{\min} \leftarrow \min\{\kappa_{\min}, \kappa_G(v_i, v_j)\}$;
    $i \leftarrow i + 1$;
return $\kappa_{\min}$;

Knotenzusammenhang (Esfahanian & Hakimi)

- erste Schleife:
  - Anzahl der Nicht-Nachbarn kann wieder höchstens $n - \delta - 1$ sein

  $\Rightarrow$ höchstens $n - \delta - 1$ MaxFlow-Aufrufe
- zweite Schleife: $\kappa(2\delta - \kappa - 3)/2$
- Gesamtkomplexität: $n - \delta - 1 + \kappa(2\delta - \kappa - 3)/2$

Kantenzusammenhangsalgorithmen

- Der Kantenzusammenhang λ kann in ungerichteten ungewichteten Graphen ebenfalls mit der MaxFlow-Prozedur gelöst werden.
- Ersetze dafür jede ungerichtete Kante durch zwei antiparallele gerichtete Kanten mit Kapazität 1 und berechne dann den lokalen Zusammenhang zwischen der entsprechenden Quelle s und der Senke t.
- Da das resultierende Netzwerk ein Unit Capacity Network vom Typ 1 ist, ist die Komplexität $\mathcal{O}(m \cdot \min\{m^{1/2}, n^{2/3}\})$.

Kantenzusammenhang λ

- Ein trivialer Algorithmus könnte einfach mit $n(n-1)/2$ MaxFlow-Aufrufen den lokalen Kantenzusammenhang aller Knotenpaare berechnen.
- Etwas besser wäre es, einen Knoten s festzuhalten und dann für alle anderen Knoten t die lokalen Zusammenhangszahlen $\lambda(s,t)$ zu berechnen.

  Mindestens einer dieser Knoten muss auf der anderen Seite eines MinCuts liegen. Deshalb ist das Minimum aller dieser $(n-1)$ lokalen Zusammenhangszahlen gleich dem (globalen) Kantenzusammenhang λ des Graphen.

  Gesamtkomplexität: $\mathcal{O}(nm \cdot \min\{n^{2/3}, m^{1/2}\})$

Kantenzusammenhang λ

- Der Algorithmus funktioniert auch, wenn man statt der ganzen Knotenmenge nur eine Teilmenge verwendet, die zwei Knoten s, t enthält, deren lokaler Zusammenhang $\lambda(s, t)$ gleich dem globalen Zusammenhang λ ist.
- Eine solche Teilmenge heißt λ-Covering.

$\Rightarrow$ Versuche, die Kardinalität dieser Knotenmenge zu reduzieren.

Lemma

Sei S ein Minimum Edge-Cut eines Graphen $G = (V, E)$ und sei $L, R \subset V$ eine Partition der Knotenmenge, so dass L and R durch S separiert werden. Wenn $\lambda(G) < \delta(G)$, dann besteht jede Komponente von $G - S$ aus mehr als $\delta(G)$ Knoten, d.h. es gilt $|L| > \delta(G)$ und $|R| > \delta(G)$.

Kantenzusammenhang λ

Beweis.

- Seien $l_1, \ldots, l_k$ die Elemente von L und sei $E[L] = E(G[L])$ die Menge der durch L induzierten Kanten in G.
- Es gilt:

$$\begin{aligned} \delta_G \cdot k &\leq \sum_{i=1}^{k} d_G(l_i) \\ &\leq 2 \cdot |E[L]| + |S| \\ &\leq 2 \cdot \frac{k(k-1)}{2} + |S| \\ &< k(k-1) + \delta(G) \end{aligned}$$

- Aus $\delta_G \cdot (k-1) < k(k-1)$ folgt $|L| = k > 1$ und $|L| = k > \delta_G$ (sowie $|R| > \delta(G)$).

Kantenzusammenhang λ

Folgerung

Wenn gilt $\lambda_G < \delta_G$, dann enthält jede Komponente von $G - S$ einen Knoten, der mit keiner Kante in S inzident ist.

Kantenzusammenhang λ

Lemma

Sei $\lambda_G < \delta_G$ und sei T ein Spannbaum von G. Dann enthalten alle Komponenten von $G - S$ mindestens einen Knoten, der kein Blatt in T ist. (D.h. die inneren Knoten von T bilden ein λ-Cover.)

Beweis.

- Nehmen wir an, dass es nicht so wäre (d.h. alle Knoten in L sind Blätter von T).
- Also für keine Kante von T sind beide Endknoten in L, d.h. $|L| = |S|$.
- Aus der Aussage des vorhergehenden Lemmas (Wenn $\lambda_G < \delta_G$, dann besteht jede Komponente von $G - S$ aus mehr als δ_G Knoten, d.h. es gilt $|L| > \delta_G$ und $|R| > \delta_G$.) folgt sofort, dass $\lambda_G = |S| = |L| > \delta_G$ (Widerspruch zur Annahme).

Spannbaum-Berechnung (Esfahanian & Hakimi)

Algorithmus von Esfahanian & Hakimi:

- Berechne Spannbaum des gegebenen Graphen.
- Wähle beliebigen inneren Knoten v des Baums.
- Berechne für jeden anderen Knoten w, der kein Blatt ist, den lokalen Kantenzusammenhang $\lambda(v, w)$.
- Das Minimum dieser Wertemenge, zusammen mit δ_G, ergibt genau den Kantenzusammenhang λ_G.
- Ein Baum mit möglichst vielen Blättern wäre vorteilhaft, aber die Konstruktion eines optimalen Baums ist $\mathcal{NP}$-hart.

Spannbaum-Berechnung (Esfahanian & Hakimi)

Esfahanian & Hakimi:

- Algorithmus zur Berechnung eines Spannbaums T von G, so dass beide Mengen, L und R eines minimalen Kantenseparators mindestens ein Blatt von T enthalten, und nach dem letzten Lemma mindestens einen inneren Knoten.

Spannbaum-Berechnung (Esfahanian & Hakimi)

Algorithmus 12 : Spannbaum-Berechnung (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$
Output : Spannbaum T mit einem Blatt und einem inneren Knoten in L bzw. R

Wähle $v \in V$;
$T \leftarrow$ alle Kanten inzident mit v;
while $|E(T)| < n - 1$ **do**
  Wähle ein Blatt w in T, so dass für alle Blätter r in T gilt:
  $|N(w) \cap (V - V(T))| \geq |N(r) \cap (V - V(T))|$;
  $T \leftarrow T \cup \{(w, x) \in E : x \in (V - V(T))\}$
return T;

Kantenzusammenhang λ (Esfahanian & Hakimi)

Algorithmus 13 : Kantenzusammenhang λ (Esfahanian & Hakimi)

Input : Ungerichteter Graph $G = (V, E)$
Output : $\lambda(G)$

Konstruiere einen Spannbaum T (voriger Algorithmus);
Sei P die kleinere der beiden Mengen
(entweder die Blätter oder die inneren Knoten von T;
Wähle einen Knoten $u \in P$;
$c \leftarrow \min\{\lambda_G(u, v) : v \in P \setminus \{u\}\}$;
$\lambda \leftarrow \min(\delta(G), c)$;
return λ;

Kantenzusammenhang λ (Esfahanian & Hakimi)

- Da P als kleinere der beiden Mengen (Blätter / innere Knoten) gewählt wird, benötigt der Algorithmus höchstens $n/2$ Berechnungen für lokale Zusammenhangszahlen.

$\Rightarrow$ Gesamtzeitkomplexität: $\mathcal{O}(\lambda mn)$

Kantenzusammenhang λ (Matula)

Definition

Ein Dominating Set für einen Graphen $G = (V, E)$ ist eine Knotenteilmenge V' von V, so dass jeder Knoten, der nicht in V' ist, mit mindestens einem Knoten in V' über eine Kante verbunden ist.

Lemma

Für den Fall, das gilt $\lambda(G) < \delta(G)$, ist jedes Dominating Set von G auch ein λ-covering von G.

Kantenzusammenhang λ (Matula)

Verbesserter Algorithmus von Matula:

- Analog zur Spannbaum-Methode, wird λ hier berechnet, indem man
  - ein Dominating Set D von G berechnet,
  - einen beliebigen Knoten $u \in D$ auswählt und
  - den lokalen Kantenzusammenhang $\lambda(u, v)$ für alle anderen Knoten $v \in D \setminus \{u\}$ berechnet.
- Das Minimum der Wertemenge (wieder zusammen mit δ_G) ist der Kantenzusammenhang.
- Obwohl die Berechnung eines Dominating Sets minimaler Kardinalität $\mathcal{NP}$-hart ist, kann man zeigen, dass der Algorithmus in Zeit $\mathcal{O}(mn)$ läuft, wenn man das Dominating Set wie im folgenden Algorithmus konstruiert.

Kantenzusammenhang λ (Matula)

Algorithmus 14 : Berechnung eines Dominating Sets (Matula)

Input : Ungerichteter Graph $G = (V, E)$
Output : Dominating Set D

Wähle $v \in V$;
$D \leftarrow \{v\}$;
while $V \setminus (D \cup N(D)) \neq \emptyset$ **do**
  Wähle einen Knoten $w \in V \setminus (D \cup N(D))$;
  $D \leftarrow D \cup \{w\}$;
return D;

k-Kanten-Zusammenhangskomponenten

David W. Matula: k-Components, Clusters, and Slicings in Graphs

- gegeben: ungewichteter, ungerichteter Graph $G = (V, E)$
- Wdh.: Eine k-Kanten-(Zusammenhangs-)Komponente von G ist ein maximaler k-kanten-zusammenhängender Teilgraph von G.
- Da wir in diesem Abschnitt nur über Kanten-Zusammenhangskomponenten sprechen, werden wir diese oft einfach als k-Komponenten bezeichnen.
- Graphen bzw. Komponenten bestehend aus einem einzelnen Knoten nennen wir *trivial*.

Vereinigung von Teilgraphen

Lemma

Seien $G_1, G_2, \ldots, G_n$ Teilgraphen von G, so dass $\bigcup_{i=1}^{n} G_i$ zusammenhängend ist, dann gilt:

$$\lambda\left(\bigcup_{i=1}^{n} G_i\right) \geq \min_{1 \leq i \leq n} \{\lambda(G_i)\}$$

Vereinigung von Teilgraphen

Beweis.

- Wenn $\bigcup_{i=1}^{n} G_i$ aus einem einzigen Knoten besteht, gilt die Behauptung (denn beide Seiten sind Null).
- Sei anderenfalls $C = (A, \bar{A})$ ein MinCut von $\bigcup_{i=1}^{n} G_i$.
- Dieser MinCut muss mindestens eine Kante enthalten, da $\bigcup_{i=1}^{n} G_i$ zusammenhängend und nicht trivial ist.
- Falls C eine Kante eines Teilgraphen G_j enthält, enthält sowohl A als auch $\bar{A}$ jeweils mindestens einen Knoten aus $V(G_j)$, d.h. C muss einen Cut für G_j enthalten.

$\Rightarrow$ Für mindestens ein $j \in \{1, \ldots, n\}$ gilt: $\lambda\left(\bigcup_{i=1}^{n} G_i\right) \geq \lambda(G_j)$

□

Vereinigung von Teilgraphen

- Für die Teilgraphen $G_1, G_2, \ldots, G_n$ von G muss jeder Cut von $G\left[\bigcup_{i=1}^{n} V(G_i)\right]$ einen Cut von $\bigcup_{i=1}^{n} G_i$ enthalten, so dass gilt:

$$\lambda\left(G\left[\bigcup_{i=1}^{n} V(G_i)\right]\right) \geq \lambda\left(\bigcup_{i=1}^{n} G_i\right)$$

Folgerung

Wenn $G_1, G_2, \ldots, G_n$ Teilgraphen von G sind, so dass $\bigcup_{i=1}^{n} G_i$ zusammenhängend ist, dann gilt:

$$\lambda\left(G\left[\bigcup_{i=1}^{n} V(G_i)\right]\right) \geq \min_{1 \leq i \leq n}\{\lambda(G_i)\}$$

Vereinigung von Teilgraphen

- Man beachte, dass in der Folgerung die Bedingung, dass $\bigcup_{i=1}^{n} G_i$ zusammenhängend ist, nicht durch die abgeschwächte Forderung, dass $G\left[\bigcup_{i=1}^{n} V(G_i)\right]$ zusammenhängend ist, ersetzt werden kann.

  Bsp.: wenn G_1 und G_2 zwei disjunkte Kreise sind, die in G durch eine einzelne Kante verbunden sind, ist die Ungleichung in der Folgerung nicht erfüllt.
- Eine weitere Konsequenz des Lemmas ist die Tatsache, dass die k-Kanten-Komponenten jedes Graphen disjunkt sind.
  (Beweis siehe Abschnitt 'Fundamentale Sätze' am Anfang des Kapitels)

Die Kohäsion / Zusammenhangsfunktion

Definition

Für jedes Element (Knoten oder Kante) $x \in V(G) \cup E(G)$ eines Graphen G ist die Kohäsion (cohesiveness) bzw. die Zusammenhangsfunktion $h(x)$ definiert als maximaler Wert des Kantenzusammenhangs von allen Teilgraphen von G, die x enthalten.

Das Maximum $\sigma(G)$ aller Kohäsionswerte des Graphen G wird als Stärke (strength) des Graphen bezeichnet, also
$\sigma(G) = \max\{\lambda(G') : G' \text{ ist Teilgraph von } G\}$.

Die Kohäsionsmatrix

- Die Zusammenhangsfunktion kann durch die (symmetrische) Kohäsionsmatrix dargestellt werden.
- Zeilen und Spalten werden durch die Knoten des Graphen indiziert, und der Eintrag für Position v_i, v_j ist jeweils die Kohäsion der Kante $\{v_i, v_j\}$, falls sie existiert, und sonst Null.
- Die Kohäsion eines Knotens ist dann das Maximum der entsprechenden Zeile oder Spalte.
- Die Stärke von G ist das Maximum aller Matrixeinträge.

Die Kohäsionsmatrix

- Für $v \in V(G)$ gilt: $0 \leq h(v) \leq \deg(v)$.
- Falls $\{v_i, v_j\} \in E(G)$, gilt: $1 \leq h(\{v_i, v_j\}) \leq \min\{\deg(v_i), \deg(v_j)\}$
- $h(x) = 0$ gilt also nur, wenn x ein isolierter Knoten ist.
- $\sigma(G) = 0$ gilt nur, wenn G keine Kante enthält.
- $\sigma(G) = 1$ gilt genau dann, wenn G ein Wald mit mindestens einer Kante ist, denn jeder Kreis würde $\sigma(G) \geq 2$ implizieren.

Eindeutige Komponentenzuordnung

- Für $x \in V(G) \cup E(G)$ und $h(x) \geq 1$ muss es für jedes $k \in \{1, \ldots, h(x)\}$ einen k-kanten-zusammenhängenden Teilgraph geben, der x enthält.
- Insbesondere muss es auch einen maximalen solchen Teilgraph (also eine k-Kanten-Komponente) geben.
- Da die k-Kanten-Komponenten sich nicht überschneiden ist dieser maximale Teilgraph (die Komponente) eindeutig.

Folgerung

Für jeden Graphen G, jedes Element $x \in V(G) \cup E(G)$ und jede Zusammenhangszahl $k \in \{1, \ldots, h(x)\}$ existiert eine eindeutige k-Kanten-Zusammenhangskomponente in G, die x enthält.

Eindeutige Komponentenzuordnung

- Die eindeutige $h(x)$-Komponente, die x enthält, hat unter allen Maximum k-kantenzusammenhängenden Teilgraphen von G, die x enthalten, die größte Knotenanzahl und heißt $h(x)$-Komponente selektiert durch x (mit Symbol H_x).
- Die Kohäsion eines Elements kann aus dem Wissen über einen beliebigen Teilgraph maximalen Kantenzusammenhangs, der das Element enthält, abgeleitet werden.
- Aus der Kenntnis der k-Kanten-Komponenten von G für alle k kann man $h(x)$ für jedes Element x bestimmen.
- Aber man kann umgekehrt auch mit Hilfe der Zusammenhangsfunktion die Komponente H_x bestimmen.

Komponentenbestimmung

Satz

Sei x ein Element des Graphen G mit $h(x) \geq 1$. Sei M_x ein maximaler zusammenhängender Teilgraph von G, der x enthält und dessen Elemente alle Kohäsion mindestens $h(x)$ haben. Dann gilt $M_x = H_x$.

Beweis.

- Für $x \in V(G) \cup E(G)$ mit $h(x) \geq 1$ sei M_x definiert wie in dem Satz.
- Dann haben für jedes $y \in V(M_x) \cup E(M_x)$ alle Elemente von H_y Kohäsion mindestens $h(y) \geq h(x)$, so dass also gilt $H_y \cup M_x = M_x$.

Komponentenbestimmung

Beweis.

- Da jedes Element von M_x in einem H_y ist, gilt:

$$M_x = \bigcup \{H_y : y \in V(M_x) \cup E(M_x)\}$$

- Nach dem Lemma gilt daher

$$\lambda(M_x) \geq h(x)$$

- Da M_x selbst ein Teilgraph von G ist, der x enthält, gilt

$$\lambda(M_x) = h(x)$$

- Damit ist M_x ein $h(x)$-kanten-zusammenhängender Teilgraph von G und muss in der $h(x)$-Komponente selektiert durch x enthalten sein, also in H_x.

Komponentenbestimmung

Beweis.

- Da wegen $M_x = \bigcup\{H_y : y \in V(M_x) \cup E(M_x)\}$ der Teilgraph H_x in M_x enthalten sein muss, gilt $M_x = H_x$.
- Der im Satz definierte Teilgraph M_x ist damit eindeutig und kann bestimmt werden, indem man ausgehend von x alle Elemente anhängt, die von x über einen Pfad erreichbar sind, dessen Elemente alle Kohäsion mindestens $h(x)$ aufweisen.

□

Folgerung

Für jeden Graph G und eine beliebige Zahl $k \in \{1, \ldots, \sigma(G)\}$ bilden die Knoten und Kanten von G mit Kohäsion mindestens k einen Graph, dessen Komponenten die k-Komponenten von G sind.

Komponentenbestimmung

- Für jeden Graph G sind die $h(x)$-Komponenten selektiert durch $x \in V(G) \cup E(G)$ von besonderem Interesse. Es wird nun gezeigt, dass in dieser Menge alle k-Kanten-Komponenten von G (für alle $k \in \{1, \ldots, \sigma(G)\}$) enthalten sind.

Folgerung

Wenn G' eine k-Komponente (für ein $k \geq 1$) des Graphen G ist, dann gibt es ein $x \in V(G) \cup E(G)$, so dass $G' = H_x$ ist.

Komponentenbestimmung

Beweis.

- Sei G' eine k-Komponente von G.
- Dann gilt $1 \leq k \leq \lambda(G')$ und G' ist damit auch eine $\lambda(G')$-Komponente von G.
- Wähle $x \in V(G') \cup E(G')$ so, dass $h(x)$ minimal ist und sei M_x definiert wie im Satz. (Man beachte, dass G' in M_x als Teilgraph enthalten sein muss.)
- $M_x = H_x$ ist eine $h(x)$-Komponente und deshalb k-kanten-zusammenhängend (da $k \leq \lambda(G') \leq h(x)$).
- Da G' ein maximaler k-kanten-zusammenhängender Teilgraph ist, gilt $G' = H_x$.

Zusammenhangsvielfalt

Definition

Für einen Graphen G sei die Zusammenhangs(komponenten)vielfalt $\eta(G)$ definiert als

$$\eta(G) = |\{H : H \text{ ist eine } k\text{-Komponente von } G \text{ für ein } k \geq 1\}|$$

- Aus dem vorangegangenen Korollar folgt $\eta(G) \leq |V(G)| + |E(G)|$.
- Man kann die Schranke aber noch genauer angeben:

Satz

Für jeden Graph G gilt:

$$\eta(G) \leq \left\lfloor \frac{|V|}{2} \right\rfloor$$

Zusammenhangsvielfalt

Beweis.

- Um den Satz zu beweisen, wird folgende Ungleichung gezeigt:

$$\eta(G) \leq \lfloor [|V(G)| + 1 - \lambda(G)]/2 \rfloor$$

- Die Ungleichung ist klar für zusammenhängende Graphen auf ein oder zwei Knoten.
- Induktion: angenommen G ist ein zusammenhängender Graph auf $n \geq 3$ Knoten und die Ungleichung gilt für zusammenhängende Graphen auf weniger als n Knoten.
- Falls $\lambda(G) = \sigma(G)$, dann gilt $\eta(G) = 1$ und somit auch die Ungleichung.
- Anderenfalls sei G' der Teilgraph von G, den man aus G durch Löschen der Elemente mit Kohäsion $\lambda(G)$ erhält, d.h. G' ist ein Graph, dessen Komponenten die $(\lambda(G) + 1)$-Komponenten von G sind.

Zusammenhangsvielfalt

Beweis.

- Da die Kanten von G mit Kohäsion $\lambda(G)$ einen Cut von G beinhalten müssen, ist entweder G' nicht zusammenhängend oder G' hat weniger Knoten als G.
- In beiden Fällen müssen die Komponenten $G'_1, G'_2, \ldots, G'_j$ von G' alle weniger Knoten als G haben. Die Ungleichung lässt sich also per Induktionsvoraussetzung auf alle G'_i mit $i \in \{1, \ldots, j\}$ anwenden.
- Ebenso gilt $\lambda(G'_i) \geq \lambda(G) + 1$ für $i \in \{1, \ldots, j\}$.
- Eine k-Komponente von G' muss nun eine k-Komponente einer Komponente von G' sein und umgekehrt.

Zusammenhangsvielfalt

Beweis.

- Deshalb gilt

$$\begin{aligned}\eta(G') = \sum_{i=1}^{j} \eta(G'_i) &\leq \sum_{i=1}^{j} \lfloor [|V(G'_i)| + 1 - \lambda(G'_i)]/2 \rfloor \\ &\leq \lfloor [|V(G')| - j\lambda(G)]/2 \rfloor\end{aligned}$$

- Jetzt gilt entweder $|V(G')| \leq |V(G)| - 1$ oder G' ist nicht zusammenhängend, so dass $j \geq 2$ und $j\lambda(G) \geq \lambda(G) + 1$.
- In jedem Fall gilt:

$$\eta(G') \leq \lfloor [|V(G)| - 1 - \lambda(G)]/2 \rfloor$$

- G selbst ist eine k-Komponente von G für $k = \lambda(G)$, und $\eta(G) = \eta(G') + 1$.

Zusammenhangsvielfalt

Beweis.

- Damit gilt:
$$\eta(G) \leq \lfloor [|V(G)| + 1 - \lambda(G)]/2 \rfloor$$
- Ungleichung aus dem Satz:
  - Die Ungleichung aus dem Satz ist für triviale Graphen klar.
  - Für nichttriviale zusammenhängende Graphen folgt sie aus der verschärften Form.
  - Für beliebige Graphen folgt sie aus der Summation über die nichttrivialen Komponenten.

Cluster und Subcluster

- Die Zusammenhangsfunktion eines Graphen G hat ein Plateau über jeder $\sigma(G)$-Komponente und evt. auch noch an anderen Stellen.
- Diese Plateaus markieren bestimmte Gebiete von (lokal) optimalem Zusammenhang. Es sind k-Komponenten, die völlig disjunkt zu $(k+1)$-Komponenten sind.

Definition

Jeder isolierte Knoten und für $k \geq 1$ jede k-Komponente von G, die keine $(k+1)$-Komponente enthält, ist ein Cluster von G.
Der Graph G ist ein Cluster, falls $\sigma(G) = \lambda(G)$.

Cluster und Subcluster

- Es ist klar, dass verschiedene Cluster keine gemeinsamen Knoten enthalten können (folgt aus Disjunktheit von k-Komponenten).
- Die Cluster von G lassen sich aus der Zusammenhangsfunktion bestimmen.

Definition

Ein induzierter Teilgraph $K[A]$ eines Clusters K des Graphen G mit $\lambda(K[A]) = \lambda(K)$ heißt Subcluster von G.

Subcluster repräsentieren also induzierte Teilgraphen von lokal maximalem Kantenzusammenhang, die nicht unbedingt inklusions-maximal sind.

Cluster und Subcluster

- Wenn $G[A]$ und $G[B]$ Subcluster von G sind, für die gilt $A \cap B = \emptyset$, dann ist $A \cup B$ ein Subcluster von G.
- Die Subcluster eines Graphen bilden unter der Relation "ist echter Teilgraph von" eine partielle Ordnung mit Clustern als maximalen Elementen.
- Die partielle Ordnung der Subcluster in einem bestimmten Cluster ist ein beschränkter Verband.
- Da alle Cluster eines Graphen disjunkt sind, ist die vollständige partielle Ordnung eine Vereinigung von disjunkten beschränkten Verbänden.

Schnitt von Subclustern

Satz

Seien $G[A]$ und $G[B]$ Subcluster eines Clusters K in G, so dass $A \cap B \neq \emptyset$. Wenn es einen MinCut von $G[A] \cup G[B]$ gibt, der die Knoten in $A - B$ von den Knoten in $B - A$ separiert und der mindestens eine Kante aus $G[A \cap B]$ enthält, dann ist $G[A \cap B]$ ein Subcluster von K in G.

Beweis.

siehe D. Matula, SIAM J. Appl. Math. 22(3), 1972. □

Slicings

Definition

Die geordnete Partition der Kanten des Graphen G, $Z = (C_1, C_2, \ldots, C_m)$, ist ein Slicing von G falls für jedes Element C_i gilt:

$$C_i \text{ ist ein Cut } (A_i, \bar{A}_i) \text{ von } \begin{cases} G & \text{für } i = 1 \\ G - \bigcup_{j=1}^{i-1} C_j & \text{für } i \in \{2, \ldots, m\} \end{cases}$$

Ein Element C_i des Slicings heißt auch *Cut des Slicings*.
Ein Slicing $Z = (C_1, C_2, \ldots, C_m)$ von G, für das es keine echte Unterpartition gibt, die ein Slicing von G ist, ist ein *minimales Slicing* von G. (Jeder Cut C_i eines minimalen Slicings muss ein minimaler Cut einer Komponente von $G - \bigcup_{j=1}^{i-1} C_j$ sein.)

Slicings

Definition

Z ist ein *Narrow Slicing* von G, falls jeder Cut C_i ein MinCut einer Komponente von $G - \bigcup_{j=1}^{i-1} C_j$ ist.

Dynamische Interpretation:

- Sequenz von nicht-leeren Cuts, die G in isolierte Knoten teilen
- Bei Minimalen/Narrow Slicings werden werden nur Minimale/Minimum Cuts in jedem Schritt verwendet.
- Jedes Narrow Slicing ist auch ein Minimales Slicing. (Das gilt umgekehrt nicht.)

Slicings

- Es ist nützlich, ein Slicing als sukzessive Zerlegung der Knotenmenge zu betrachten.
- Der i-te Cut $C_i = (A_i, \bar{A}_i)_i$ des Slicings Z von G ist ein Cut von $G - \bigcup_{j=1}^{i-1} C_j$.
- Da $A_i \cup \bar{A}_i = V(G - \bigcup_{j=1}^{i-1} C_j) = V(G)$, betrifft $(A_i, \bar{A}_i)_i$ die gleiche Partition der Knoten von G wie der Cut $(A_i, \bar{A}_i)$ von G.
- Es gilt

$$(A_i, \bar{A}_i)_i = (A_i, \bar{A}_i) - \bigcup_{j=1}^{i-1} C_j$$

- Also enthält $(A_i, \bar{A}_i)_i$ nicht unbedingt alle Kanten des Cuts $(A_i, \bar{A}_i)$ von G für $i > 1$.
- Man beachte, dass die Kantenmenge $(A_i, \bar{A}_i)$ implizit von G abhängt und dass die die Kantenmenge $(A_i, \bar{A}_i)_i$ implizit von G und Z abhängt. Die Knotenpartition ist jedoch explizit und in beiden Fällen gleich!

Knotenpartitionen

- Die Repräsentation der Cuts des Slicings Z als Knotenpartition von $V(G)$ bietet eine nützliche Charakterisierung von jedem Graphen $G - \bigcup_{j=1}^{i-1} C_j$ im Sinne einer Vereinigung von induzierten Teilgraphen von G.
- Da C_1 die Knotenmenge A_1 von $\bar{A}_1$ separiert, und damit $G - C_1 = G[A_1] \cup G[\bar{A}_1]$, gilt somit:

$$G - (C_1 \cup C_2) = G[A_1 \cap A_2] \cup G[A_1 \cap \bar{A}_2] \cup G[\bar{A}_1 \cap A_2] \cup G[\bar{A}_1 \cap \bar{A}_2]$$

Satz

Induktiv folgt: Sei $Z = (C_1, C_2, \ldots, C_m)$ ein Slicing von G mit $C_i = (A_i, \bar{A}_i)_i$ für $i \in \{1, \ldots, m\}$. Dann gilt für $j \in \{1, \ldots, m\}$:

$$G - \bigcup_{k=1}^{j} C_k = \bigcup \left\{ G \left[\bigcap_{i \in s} A_i \cap \bigcap_{i \notin s} \bar{A}_i \right] : s \subset \{1, 2, \ldots, j\} \right\}$$

Knotenpartitionen

- Jeder Graph G kann geschrieben werden als $G = G[P_1] \cup G[P_2] \cup \ldots \cup G[P_n]$, wobei jedes $G[P_i]$ eine Komponente von G ist.
- Sei dann $\{P_1, P_2, \ldots, P_n\}$ die Komponenten-Knoten-Partition von G.
- Falls G nicht zusammenhängend ist, können die induzierten Teilgraphen auf der rechten Seite im letzten Satz unzusammenhängend sein.
- Sei $Z = (C_1, C_2, \ldots, C_m)$ ein Slicing von G, wobei G die Komponenten-Knoten-Partition $\{P_1, P_2, \ldots, P_n\}$ hat. Dann hat $G - \bigcup_{k=1}^{j} C_k$ die Komponenten-Knoten-Partition $G - \bigcup_{k=1}^{j} C_k =$

$$\bigcup \left\{ G\left[P_l \cap \bigcap_{i \in s} A_i \cap \bigcap_{i \notin s} \bar{A}_i \right] : 1 \leq l \leq n, s \subset \{1, 2, \ldots, j\} \right\}$$

Knotenpartitionen

- Man beachte, dass die Komponenten-Knoten-Partition von $G - \bigcup_{k=1}^{j} C_k$ eine Subpartition der Komponenten-Knoten-Partition von $G - \bigcup_{k=1}^{j-1} C_k$ für $j \in \{1, \ldots, m\}$ ist.
- D.h., das Slicing $Z = (C_1, C_2, \ldots, C_m)$ bewirkt eine geschachtelte Sequenz von $m + 1$ Knotensubpartitionen von der Komponenten-Knoten-Partition $\{P_1, P_2, \ldots, P_n\}$ bis hinunter zur minimalen Partition bestehend aus lauter einzelnen Knoten.
- Ein Cut C_i des Slicings $Z = (C_1, C_2, \ldots, C_m)$ kann einige der Komponenten von $G - \bigcup_{j=1}^{i-1} C_j$ intakt lassen. Es ist daher sinnvoll zu spezifizieren, welche Komponenten von C_i wirklich zertrennt werden.

Knotenpartitionen

- Die Subgraphen $G_1, G_2, \ldots, G_m$ von G sind die *durch das Slicing* $Z = (C_1, C_2, \ldots, C_m)$ *zertrennten Teilgraphen*, wenn jedes G_i genau die Komponenten von $G - \bigcup_{j=1}^{i-1} C_j$ enthält, deren Knoten durch C_i separiert werden.
- Jeder vom Slicing Z zertrennte Teilgraph G_i ist dann eine Vereinigung von induzierten Teilgraphen von G.
- Da ein minimales Slicing nur sukzessive Cuts von individuellen Komponenten beinhaltet, gilt das folgende Korollar.

Knotenpartitionen

Folgerung

Die durch ein minimales Slicing $Z = (C_1, C_2, \ldots, C_m)$ zertrennten Teilgraphen sind alle zusammenhängende induzierte Teilgraphen von G.

- D.h., die geschachtelte Sequenz von Komponenten-Knoten-Partitionen für ein minimales Slicing ist derart, dass jede Subpartition aus der vorhergehenden durch Aufspaltung von genau einem Teil hervorgeht, und entspricht demzufolge einer Maximalen Kette im Verband der Komponenten-Knoten-Partitionen.

Länge von Slicings

Definition

Sei die Länge $\ell(Z)$ eines Slicings Z des Graphen G die Anzahl der Cuts des Slicings (also die Kardinalität der Kantenpartition Z)

- Jeder Cut des Slicings erhöht die Anzahl der Zusammenhangskomponenten im Graphen.
- Diese Erhöhung ist genau dann immer gleich Eins, wenn die Cuts minimal sind (bzw. das Slicing ein minimales Slicing ist).

Satz

Für jeden Graphen G mit $|E(G)| \geq 1$ gilt:
$\max\{\ell(Z) | Z$ *ist ein Slicing von* $G\} = |V(G)| - \#Komponenten(G)$ *und dieses Maximum wird genau von den minimalen Slicings erreicht. (Die Größe rechts heißt auch Cocycle-Rang von* G*.)*

Länge von Slicings

Definition

Ein Graph ist k-partit, wenn die Knotenmenge $V(G)$ in k Mengen $V_1, \ldots, V_k$ partitioniert werden kann, so dass jeder induzierte Graph $G[V_i]$ ($i \in \{1, \ldots, k\}$) keine Kante enthält.

- Die Länge eines Slicings Z von einem Graphen G kann Eins sein, nämlich dann, wenn G bipartit ist.
- Allgemein hängt der minimale Wert für die Länge eines Slicings eines Graphen mit der minimalen Zahl k zusammen, so dass G ein k-partiter Graph ist.
- Die minimale Anzahl k, so dass G k-partit ist, heißt chromatische Zahl $\chi(G)$ des Graphen G.

Länge von Slicings

Satz

Für jeden Graph G mit $|E(G)| \geq 1$ gilt:

$$\min\{\ell(Z) : Z \text{ ist ein Slicing von } G\} = \lceil \log_2 \chi(G) \rceil\}$$

Beweis.

siehe D. Matula, SIAM J. Appl. Math. 22(3), 1972. □

Weite von Slicings

Definition

Die Weite $w(Z)$ eines Slicings Z des Graphen G sei definiert durch

$$w(Z) = \max\{|C| : c \text{ ist ein Cut von } G\}$$

und jeder Cut C von Z mit $|C| = w(Z)$ ist ein Wide Cut des Slicings Z.

- Man beachte, dass ein MinCut sich auf den Graph bezieht, während ein Wide Cut sich auf ein bestimmtes Slicing bezieht.

Weite von Slicings

- Da jeder Cut $C_i = (A_i, \bar{A}_i)_i$ eines Slicings Z von G in einem Cut $(A_i, \bar{A}_i)$ von G enthalten ist, und da jedes Slicing mit einem beliebigem Cut von G beginnen kann, muss die maximale Weite eines Cuts gleich der maximalen Anzahl von Kanten in einem beliebigen Cut sein.

Satz

Für jeden Graphen mit $|E(G)| \geq 1$ *gilt:*

$$\max\{w(Z) : Z \text{ ist Slicing von } G\} = \max\{|C| : C \text{ ist Cut von } G\}$$

Weite von Slicings

- Die minimale Weite eines Slicings ist ähnlich verwandt zu MinCuts, aber nicht vom ganzen Graphen G.

Satz

Für jeden Graphen mit $|E(G)| \geq 1$ *gilt:*

$$\min\{w(Z) : Z \text{ ist Slicing von } G\} = \sigma(G)$$

- Es ergibt sich damit folgende Dualität zwischen Slicings und Subgraphen:
  $\min_Z \max_C \{|C| : C \text{ ist Cut des Slicings } Z \text{ von } G\} =$
  $\max_{G'} \min_C \{|C| : C \text{ ist Cut des Teilgraphen } G' \text{ von } G\}$

Weite von Slicings

Beweis.

$\min\{w(Z) : Z \text{ ist Slicing von } G\} \geq$
$\max\{\lambda(G') : G' \text{ ist Teilgraph von } G\} = \sigma(G)$:

- Sei K ein Cluster von G mit $\lambda(K) = \sigma(K)$.
- Dann gilt für jedes Slicing Z von G, dass es einen ersten Cut C_i gibt, der Knoten von K separiert.
- Dann muss C_i einen Cut für K enthalten, so dass $|C_i| \geq \lambda(K) = \sigma(G)$.
- Also gilt $w(Z) \geq \sigma(Z)$ für jedes Slicing Z.

Weite von Slicings

Beweis.

$\min\{w(Z) : Z \text{ ist Slicing von } G\} \leq$
$\sigma(G) = \max\{\lambda(G') : G' \text{ ist Teilgraph von } G\} = \sigma(G)$:

- Sei $Z^* = (C_1, \ldots, C_m)$ ein Narrow Slicing von G mit Wide Cut C_i.
- Sei G_i der Teilgraph von G, der durch C_i zertrennt wird.
- Da ein Narrow Slicing nur Minimum Cuts benutzt, folgt $\lambda(G_i) = w(Z^*)$.

Weite von Slicings

- Die Weite eines Slicings muss größer oder gleich der durchschnittlichen Anzahl der Kanten in den Cuts des Slicings sein.
- Aus dem letzten Satz und dem Satz über die maximale Länge eines Slicings folgt damit die folgende untere Schranke für die Stärke $\sigma(G)$:

Folgerung

Für jeden Graphen mit $|E(G)| \geq 1$ *gilt:*

$$\sigma(G) \geq |E(G)|/(|V(G)| - 1) > |E(G)|/|V(G)|$$

Narrow Slicings

- Im Beweis des letzten Satzes war jedes Narrow Slicing ausreichend, um die Ungleichung der zweiten Richtung zu beweisen. Deshalb gelten folgende Korollare:

Folgerung

Für jedes Narrow Slicing Z^* von G gilt: $w(Z^*) = \sigma(G)$.

Folgerung

Jeder Wide Cut eines Narrow Slicings von G ist ein Minimum Cut eines Subclusters von G.

- Die nächsten zwei Folgerungen zeigen, dass man ein Narrow Slicing benutzen kann, um die k-Komponenten ($k \geq 1$) und die Zusammenhangsfunktion zu berechnen.

Narrow Slicings und k-Komponenten

Folgerung

Seien $G[A_1]$, $G[A_2]$,...,$G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \ldots, C_m)$ getrennt werden.
Dann ist jede k-Komponente von G gleich einem $G[A_i]$ für ein $i \in \{1, \ldots, m\}$.
Weiterhin ist ein $G[A_i]$ ($i \in \{1, \ldots, m\}$) genau dann eine k-Komponente von G, wenn

$$A_j \supset A_i \text{ für } j < i \Rightarrow |C_j| < |C_i|.$$

Und solch ein $G[A_i]$ ist genau dann ein Cluster von G, wenn auch gilt

$$A_j \subset A_i \text{ für } j > i \Rightarrow |C_j| \leq |C_i|.$$

Narrow Slicings und k-Komponenten

Beweis.

- Seien $G[A_1],\ldots,G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \ldots, C_m)$ getrennt werden, so dass $\lambda(G[A_i]) = |C_i|$ $(i \in \{1,\ldots,m\})$.
- Sei H eine k-Komponente von G für ein $k \in \{1,\ldots,\sigma(G)\}$.
- Dann muss der erste Cut C_n aus Z, der Knoten aus H separiert, einen Cut von H enthalten, also $|C_n| = \lambda(G[A_n]) \geq \lambda(H)$.
- Da H zusammenhängend ist, muss H ein Teilgraph von $G[A_n]$, weil C_n der erste Cut ist, der Knoten aus H separiert.
- Daraus folgt, dass $G[A_n]$ ein $\lambda(H)$-kanten-zusammenhängender Graph ist $(\lambda(H) \leq \lambda(G[A_n]))$.
- Da H ein maximaler $\lambda(H)$-kanten-zusammenhängender Graph (per Def.) sowie ein Teilgraph von $G[A_n]$ ist, gilt $H = G[A_n]$.

Narrow Slicings und k-Komponenten

Beweis.

- $G[A_i]$ $(i \in \{1, \ldots, m\})$ ist in einer $\lambda(G[A_i])$-Komponente H von G enthalten, und aufgrund des vorangegangenen Arguments gilt $H = G[A_i]$ für ein $j \leq i$.
- Also ist $G[A_i]$ selbst eine $\lambda(G[A_i])$-Komponente von G genau dann, wenn $A_j \supset A_i$ für $j < i$ impliziert, dass $|C_j| < |C_i|$.
- Falls weiterhin $G[A_i]$ eine $\lambda(G[A_i])$-Komponente ist, dann handelt es sich genau dann um ein Cluster, wenn $A_j \subset A_i$ für $j > i$ impliziert, dass $|C_j| \leq |C_i|$.
- Also enthalten die Teilgraphen, die durch ein beliebiges Narrow Slicing von G zertrennt werden, alle k-Komponenten von G, und damit alle Cluster von G, die keine isolierten Knoten sind.

Narrow Slicings und Kohäsionsfunktion

Folgerung

Seien $G[A_1]$, $G[A_2]$,..., $G[A_m]$ die Teilgraphen, die durch das Narrow Slicing $Z = (C_1, C_2, \ldots, C_m)$ getrennt werden.
Dann gilt für jedes Graphelement $x \in V(G) \cup E(G)$:

$$h(x) = \begin{cases} 0, \text{ falls } x \text{ ein isolierter Knoten in } G \text{ ist,} \\ \max_i\{|C_i| : x \in A_i \cup E(G[A_i])\}, \text{ sonst} \end{cases}$$

- Wenn $G[A_1], \ldots, G[A_m]$ die Subgraphen sind, die durch ein Narrow Slicing $Z = (C_1, \ldots, C_m)$ von G zertrennt werden, dann müssen nicht alle m Subgraphen $G[A_i]$ k-Komponenten sein.
- Insbesondere, wenn $G[A_i]$ ein Cluster mit $\lambda(G[A_i]) \geq 2$ ist, dann sind mindestens $\lambda(G[A_i]) - 1$ der Teilgraphen $G[A_j]$ mit $j > i$ echte Teilgraphen von $G[A_i]$ und können demzufolge keine k-Komponenten von G für ein $k \geq 1$ sein.

Berechnung der Zusammenhangsfunktion

- Der Algorithmus basiert auf der sequentiellen Bestimmung von globalen Minimum Cuts für höchstens $|V(G)| - 1$ induzierte Teilgraphen von G.

Narrow Slicing Algorithmus

Algorithmus 15 : Berechnung eines Narrow Slicings (Matula)

Input : Graph $G = (V, E)$ mit N Komponenten und mindestens einer Kante
Output : Ein Narrow Slicing Z

Repräsentiere G als Vereinigung seiner Komponenten $G = \bigcup_{j=1}^{N} G[P_{1,j}]$;

$i \leftarrow 1$;
while $i<|V(G)|-N$ **do**
  Wähle $G_i = G[P_{i,k}]$ für ein k, so dass $|P_{i,k}| \geq 2$;
  Finde einen MinCut $C_i = (A, \bar{A})$ von G_i. (Beachte $A \cup \bar{A} = V(G_i)$);
  Definiere neue Komponenten-Knoten-Partition $\{P_{i+1,j}\}$ $(j \in \{1, \ldots, N+i\})$:

$$P_{i+1,j} = \begin{cases} P_{i,j} & \text{für } 1 \leq j < k, \\ A & \text{für } j = k, \\ \bar{A} & \text{für } j = k+1, \\ P_{i,j-1} & \text{für } k+1 < j \leq N+i, \end{cases} \quad \text{mit } G - \bigcup_{n=1}^{i} C_n = \bigcup_{j=1}^{N+i} G[P_{i+1,j}];$$

  $i \leftarrow i+1$;

$G - \bigcup_{n=1}^{|V(G)|-N} C_n$ besteht jetzt nur noch aus isolierten Knoten und $Z = (C_1, \ldots, C_{|V(G)|-N})$ ist ein Narrow Slicing;
return Z;

Narrow Slicing Algorithmus

- Ursprünglich wurde von Matula vorgeschlagen, den MinCut mit Hilfe des Cut Trees von Gomory/Hu zu berechnen. Hier kann man aber auch andere Algorithmen einsetzen, z.B. den Stoer/Wagner-Algorithmus.
- Der Narrow Slicing Algorithmus bestimmt explizit ein Narrow Slicing $Z = (C_1, \ldots, C_{|V(G)|-N})$, eine Liste $G_1, \ldots, G_{|V(G)|-N}$ von induzierten Graphen, die durch das Slicing zertrennt werden, sowie die durch Z verursachte geschachtelte Sequenz von Komponenten-Knoten-Partitionen.
- Daraus können dann entsprechend den früheren Sätzen sehr einfach die Zusammenhangsfunktion, die k-Komponenten und die Cluster von G bestimmt werden.

Narrow Slicing Algorithmus

- $G_i = G[P_{i,k}]$ hat höchstens $|V(G)| + 2 - N - i$ Knoten.
- Also müssten bei Verwendung der Cut Tree Methode höchstens

$$\sum_{i=1}^{|V(G)|-N} (|V(G)| + 1 - N - i) = \frac{1}{2}(|V(G)| - N)(|V(G)| - N + 1)$$

  Flussprobleme einfacher Art gelöst werden.
- Für einen zusammenhängenden Graphen wären das höchstens $\binom{|V(G)|}{2}$ Flussprobleme.
- Matula gibt die Komplexität grob mit n^4 bis n^5 an.
- Das entspricht auch ungefähr der Größenordnung, die man bei Verwendung des Stoer/Wagner-Algorithmusses errechnen würde ($\mathcal{O}(m_i n_i + n_i^2 \log n_i)$ pro Teilproblem mit n_i Knoten und m_i Kanten, summiert für $n_i = n - 1, \ldots, 1$)

Übersicht

Ziel

- Algorithmus zur Lösung des All Pairs Shortest Paths (APSP) Problems
- löst APSP-Problem für gewichtete gerichtete Graphen, in denen die Kantengewichte ganze Zahlen mit kleinem Absolutbetrag sind, in Zeit $\tilde{\mathcal{O}}(n^{2+\mu})$
- μ erfüllt dabei die Bedingung $\omega(1,\mu,1) = 1 + 2\mu$, wobei $\omega(1,\mu,1)$ der Zeitkomplexitätsexponent der Multiplikation einer $n \times n^{\mu}$ mit einer $n^{\mu} \times n$ Matrix ist.
- Momentan beste Schranke: $\mu < 0,575$ (Coppersmith) impliziert Komplexität $\tilde{\mathcal{O}}(n^{2,575})$.

Distanzprodukt

Definition

Sei $A = (a_{ij})$ eine $n \times m$ Matrix und $B = (b_{ij})$ eine $m \times n$ Matrix.
Das min/plus- oder Distanzprodukt von A und B, symbolisch $A \star B$ ist die $n \times n$ Matrix $C = (c_{ij})$, so dass gilt:

$$c_{ij} = \min_{k=1}^{m} \{a_{ik} + b_{kj}\} \text{ für } 1 \leq i, j \leq n.$$

Distanzprodukt-Berechnung

- Das Distanzprodukt kann naiv in Zeit $\mathcal{O}(n^2 m)$ berechnet werden.
- Alon et al.:
  - Methode mit schneller Matrixmultiplikation und schneller Integermultiplikation (Schönhage-Strassen) für den Fall, dass die Werte im Bereich $\{-M, \ldots, 0, \ldots, M\} \cup \{+\infty\}$ liegen
  - Laufzeit $\tilde{\mathcal{O}}(Mn^{\omega(1,r,1)})$, wobei $m = n^r$.
  - $\mathcal{O}(n^{\omega(1,r,1)})$ ist hier die Anzahl algebraischer Operationen, die benötigt werden, um das (normale) Produkt einer $n \times n^r$ und einer $n^r \times n$ Matrix zu berechnen
- Laufzeit hängt von M ab. Für große Werte von M ist der naive (von M unabhängige) Algorithmus schneller.
- Der folgende Algorithmus benutzt die schnellere von beiden Methoden.

Algorithmus zur Distanzprodukt-Berechnung

Algorithmus 16 : `dist-prod`(A,B,M)

Input : $n \times m$ Matrix A, $m \times n$ Matrix B (wobei $m = n^r$). Alle Elemente von A und B sind ganze Zahlen, wobei alle Einträge mit Absolutwert $> M$ durch ∞ ersetzt werden.

Output : Distanzprodukt $C = A \star B$ ($n \times n$ Matrix)

if $Mn^{\omega(1,r,1)} \leq n^{2+r}$ **then**

$$a'_{ij} \leftarrow \begin{cases} (m+1)^{M-a_{ij}} & \text{wenn } |a_{ij}| \leq M \\ 0 & \text{sonst} \end{cases};$$

$$b'_{ij} \leftarrow \begin{cases} (m+1)^{M-b_{ij}} & \text{wenn } |b_{ij}| \leq M \\ 0 & \text{sonst} \end{cases};$$

$C' \leftarrow \texttt{fast-prod}(A', B')$;

$$c_{ij} \leftarrow \begin{cases} 2M - \lfloor \log_{m+1} c'_{ij} \rfloor & \text{wenn } c'_{ij} > 0 \\ +\infty & \text{sonst} \end{cases};$$

else

$c_{ij} \leftarrow \min_{k=1}^{m} \{a_{ik} + b_{kj}\} \quad (1 \leq i,j \leq n)$;

return C;

Algorithmus zur Distanzprodukt-Berechnung

Lemma

Algorithmus `dist-prod` *berechnet das Distanz-Produkt einer* $n \times n^r$ *und einer* $n^r \times n$ *Matrix, deren endliche Einträge alle Absolutbetrag höchstens* M *haben in Zeit* $\tilde{\mathcal{O}}(\min\{Mn^{\omega(1,r,1)}, n^{2+r}\})$.

Beweis.

- Wenn $n^{2+r} < Mn^{\omega(1,r,1)}$, dann berechnet `dist-prod` das Distanz-Produkt von A und B mit Hilfe des naiven Algorithmus in Zeit $\mathcal{O}(n^{2+r})$ und wir sind fertig

Algorithmus zur Distanzprodukt-Berechnung

Beweis.

- Fall $n^{2+r} \geq Mn^{\omega(1,r,1)}$:
  - Korrektheit: $\forall i, j \in \{1, \ldots, n\}$:

$$c'_{ij} = \sum_{\substack{k \in \{1, \ldots m\} \\ a_{ik} \neq \infty \\ b_{kj} \neq \infty}} (m+1)^{2M-(a_{ik}+b_{kj})}$$

$\Rightarrow$

$$c_{ij} = \min_{k=1}^{m}\{a_{ik} + b_{kj}\} = 2M - \lfloor \log_{m+1} c'_{ij} \rfloor$$

Algorithmus zur Distanzprodukt-Berechnung

Beweis.

- weiter Fall $n^{2+r} \geq Mn^{\omega(1,r,1)}$:
  - Komplexität?
  - `fast-prod` führt $\tilde{\mathcal{O}}(n^{\omega(1,r,1)})$ arithmetische Operationen auf $\mathcal{O}(M \log n)$-Bit Integern aus.
  - Um große Zwischenergebnisse zu vermeiden, werden diese Multiplikationen z.B. $\mod (m+1)^{4M+1}$ ausgeführt.
  - Der Schönhage-Strassen-Algorithmus multipliziert zwei k-Bit Integer mit $\mathcal{O}(k \log k \log\log k)$ Bit-Operationen.
  - Mit $k = \mathcal{O}(M \log n)$ erhält man für jede arithmetische Operation $\tilde{\mathcal{O}}(M \log n)$.
  - Die Logarithmen in der Berechnung von c_{ij} können mit binärer Suche implementiert werden.
  - Die Komplexität ist also $\tilde{\mathcal{O}}(Mn^{\omega(1,r,1)})$.

Zeugen für Distanz-Produkte

Definition

Sei A eine $n \times m$ Matrix und B eine $m \times n$ Matrix.
Eine $n \times n$ Matrix heißt Zeugen-Matrix für das Distanz-Produkt $C = A \star B$, wenn für alle $1 \leq i, j \leq n$ gilt, dass $1 \leq w_{ij} \leq m$ und $c_{ij} = a_{i,w_{ij}} + b_{w_{ij},j}$.

- Man kann den Algorithmus `dist-prod` so erweitern, dass er auch eine Zeugen-Matrix zurückliefert. Die Laufzeit erhöht sich dabei nur um einen polylogarithmischen Faktor.

Zeugen für Distanz-Produkte

- Eine einfache (aber teure) Möglichkeit, die Zeugen für $C = A \star B$ zu berechnen, ist:
  - A ist $n \times m$, B ist $m \times n$ Matrix
  - Seien $A' = (a'_{ij})$ und $B' = (b'_{ij})$ Matrizen, so dass gilt: $a'_{ij} = ma_{ij} + j - 1$ und $b'_{ji} = mb_{ji}$ für alle $1 \leq i \leq n$ und $1 \leq j \leq m$.
  - Wenn man nun das Distanzprodukt $C' = A' \star B'$ berechnet, dann ist $\lfloor C'/m \rfloor$ das Distanzprodukt $A \star B$ und $(C' \mod m) + 1$ ist eine korrespondierende Zeugen-Matrix.
  - Außerdem sind alle Zeugen die kleinsten möglichen Zeugen.
  - Der Nachteil ist: die Einträge von A und B werden mit m multipliziert, was eine Verlangsamung der Prozedur `dist-prod` um den Faktor m zur Folge hat (der sehr groß sein kann).

Zeugen für Distanz-Produkte

- Es gibt einen effizienteren Weg, die Zeugen zu finden.
- Zunächst für den Fall eindeutiger Zeugen:
  - Für $1 \leq k \leq m$ und $1 \leq \ell \leq \lceil \log_2 m \rceil + 1$ sei $\text{bit}_\ell(k)$ das ℓ-te Bit in der Binärrepräsentation von k (wobei $\text{bit}_1(k)$ das Least Significant Bit von k ist).
  - Sei $I_\ell = \{k : \quad 1 \leq k \leq m \quad \wedge \quad \text{bit}_\ell(k) = 1\}$.

Definition

Sei A eine $n \times m$ Matrix und sei $I \subseteq \{1, 2, \ldots, m\}$ eine Teilmenge der (Spalten-)Indizes.
Dann ist $A[*, I]$ definiert als die Matrix, die aus den Spalten von A besteht, deren Index zu I gehört.
Entsprechend ist für eine $m \times n$ Matrix B der Ausdruck $B[I, *]$ gleichbedeutend mit der Matrix, die sich aus den Zeilen von B zusammensetzt, deren Index zu I gehört.

Zeugen für Distanz-Produkte

- Fortsetzung für den Fall eindeutiger Zeugen:
  - Um alle eindeutigen Zeugen für Elemente von $C = A \star B$ zu finden, berechnet man die $\mathcal{O}(\log m)$ Distanzprodukte $C_\ell = A[*, I_\ell] \star B[I_\ell, *]$ für $1 \leq \ell \leq \lceil \log_2 m \rceil + 1$.
  - Sei $C_\ell = (c_{ij}^{(\ell)})$.
  - $c_{ij}^{(\ell)} = c_{ij}$ gilt genau dann, wenn es einen Zeugen für c_{ij} gibt, dessen ℓ-tes Bit 1 ist.
  - Falls c_{ij} einen eindeutigen Zeugen hat, dann können diese Bedingungen benutzt werden, um die einzelnen Bits in der Binärrepräsentation von w_{ij} (und damit den Wert w_{ij}) zu bestimmen.
  - Man beachte, dass man nicht wissen muss, ob c_{ij} einen eindeutigen Zeugen hat. Man rekonstruiert einen Kandidaten w_{ij} und überprüft, ob $c_{ij} = a_{i,w_{ij}} + b_{w_{ij},j}$ gilt.

Zeugen für Distanz-Produkte

- Vorgehen im Fall mehrerer Zeugen: Sampling
  - Intention:
    Wir wollen eine Teilmenge der Indizes $\{1, \ldots, m\}$ auswählen, so dass *genau ein* Index ein Zeuge für ein bestimmtes c_{ij} ist.
  - Problem: Wir wissen nicht wieviele Zeugen es gibt.
    - Wenn es wenige Zeugen gibt, brauchen wir eine große Teilmenge der Indizes, damit überhaupt ein Zeuge dabei ist.
    - Wenn es viele gibt, dürfen wir nur eine kleine Teilmenge wählen, da die Auswahl von mehr als einem Zeugen dazu führt, dass wir die Binärrepräsentation nicht rekonstruieren können.
  - Lösung:
    Wir probieren alle möglichen Größenordnungen 2^r (mit $r \in \{1, \ldots, \log m\}$) für die Anzahl der Zeugen aus.
  - Für jedes r wählt man $s = c \log n$ zufällige Teilmengen $R_{r1}, \ldots, R_{rs}$ der Größe $m/2^r$ aus der Grundmenge $\{1, \ldots, m\}$.

Zeugen für Distanz-Produkte

- Vorgehen im Fall mehrerer Zeugen (Fortsetzung):
  - Für jede solche zufällige Teilmenge R_{rt} (wobei $1 \leq r \leq \log m$ und $1 \leq t \leq s$) suchen wir eindeutige Zeugen für das Produkt $A[*, R_{rt}] \star B[R_{rt}, *]$.
  - Wenn solch ein Zeuge gefunden wird, überprüft man, ob er auch ein Zeuge für das ursprüngliche Distanzprodukt $A \star B$ ist. (Durch das Weglassen von Zeilen/Spalten könnte das urprüngliche Minimum entfernt worden sein.)
  - Wenn die Konstante c groß genug ist, dann werden auf diese Weise mit sehr hoher Wahrscheinlichkeit Zeugen für alle Positionen gefunden (hier ohne Beweis).
  - Dieses Vorgehen führt zu einem randomisierten Algorithmus für die Berechnung der Zeugen des Distanzprodukts $A \star B$, der $\mathcal{O}(\log^3 n)$ einfache Distanzprodukte von Matrizen gleicher oder kleinerer Größe benutzt.

Zeugen für Distanz-Produkte

- Dieser Algorithmus kann mit Hilfe von Ergebnissen von Alon und Naor (1996) derandomisiert werden, wobei sich die Komplexität nur um einen polylogarithmischen Faktor verschlechtert.

Lemma

Eine erweiterte Version von Algorithmus `dist-prod` *berechnet das Distanzprodukt einer $n \times n^r$ Matrix mit einer $n^r \times n$ Matrix, deren endliche Einträge alle Betrag höchstens M haben, zusammen mit einer entsprechenden Zeugenmatrix in Zeit $\tilde{\mathcal{O}}(\min\{Mn^{\omega(1,r,1)}, n^{2+r}\})$.*

Randomisierter Kürzeste-Wege-Algorithmus

Algorithmus 17 : `rand-short-path`(D)

Input : $n \times n$ Matrix D der Kantengewichte
Output : $n \times n$ Matrix F, die mit hoher Wahrscheinlichkeit, die Distanzen des Graphen enthält, sowie eine Zeugen-Matrix W

$F \leftarrow D$;
$W \leftarrow 0$;
$M \leftarrow \max\{|d_{ij}| : d_{ij} \neq +\infty\}$;
for $\ell \leftarrow 1 \ldots \lceil \log_{3/2} n \rceil$ **do**
  $s \leftarrow (3/2)^{\ell}$;
  $B \leftarrow \mathtt{rand}(\{1, \ldots, n\}, (9 \ln n)/s)$;
  $(F', W') \leftarrow \mathtt{dist-prod}(F[*, B], F[B, *], sM)$;
  foreach $1 \leq i, j \leq n$ **do**
    if $f'_{ij} < f_{ij}$ **then**
      $f_{ij} \leftarrow f'_{ij}$;
      $w_{ij} \leftarrow b_{w'_{ij}}$;
return (F, W);

Randomisierter Kürzeste-Wege-Algorithmus

- Einfacher randomisierter Algorithmus `rand-short-path` zur Bestimmung der Distanzen und einer Repräsentation kürzester Pfade in einem gerichteten Graphen mit n Knoten, in dem alle Kantengewichte aus $\{-M, \ldots, 0, \ldots, M\}$ kommen
- Annahme: $V = \{1, \ldots, n\}$
- Eingabe: $n \times n$ Matrix D mit Gewichten bzw. Kantenlängen (d_{ij} ist Länge der gerichteten Kante von i nach j, sonst ∞)
- Zunächst wird F mit D initialisiert.
- Dann werden $\lceil \log_{3/2} n \rceil$ Iterationen ausgeführt, in denen s auf $(3/2)^\ell$ gesetzt wird und dann aus der Knotenmenge eine Teilmenge B ausgewählt wird, in die jeder Knoten unabhängig mit Wahrscheinlichkeit $\min\{1, (9 \ln n)/s\}$ aufgenommen wird.
- Dann wird das Distanzprodukt $F' = F[*, B] \star F[B, *]$ (mit Betragsobergrenze sM) berechnet.

Randomisierter Kürzeste-Wege-Algorithmus

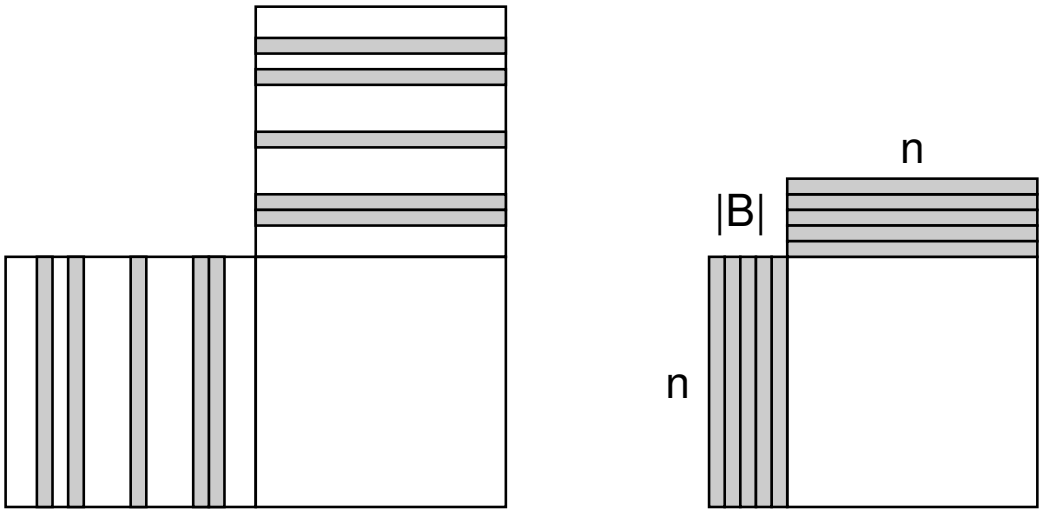


Abbildung: Ersetzung des quadratischen Produkts $F \star F$ durch das rechteckige Produkt $F[*, B] \star F[B, *]$

Randomisierter Kürzeste-Wege-Algorithmus

- Es wird dabei auch eine Zeugenmatrix W' bestimmt.
- Dann wird jeder Eintrag von F' mit dem zugehörigen Eintrag von F verglichen.
- Im Fall einer Verbesserung wird der Wert von F' nach F und der Zeuge von W' nach W übertragen.
  ($b_{w'_{ij}}$ ist das w'_{ij}-te Element von B)

Randomisierter Kürzeste-Wege-Algorithmus

- Sei $\delta(i,j)$ die (gewichtete) Distanz von i nach j.

Lemma

Zu jeder Phase gilt für Algorithmus `rand-short-path` *für alle* $i,j \in V$*:*

1. $f_{ij} \geq \delta(i,j)$.
2. *Falls* $w_{ij} = 0$*, dann ist* $f_{ij} = d_{ij}$*. Ansonsten gilt* $1 \leq w_{ij} \leq n$ *und* $f_{ij} \geq f_{i,w_{ij}} + f_{w_{ij},j}$.
3. *Falls* $\delta(i,j) = \delta(i,k) + \delta(k,j)$ *und falls am Anfang einer Iteration gilt* $f_{ik} = \delta(i,k)$, $f_{kj} = \delta(k,j)$, $|f_{ik}|, |f_{kj}| \leq sM$ *und* $k \in B$*, dann gilt am Ende der Iteration* $f_{ij} = \delta(i,j)$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

1. Die Eigenschaft gilt bei Initialisierung von F mit D. In jeder Iteration wählt der Algorithmus eine Menge B und bestimmt

$$\begin{aligned} f'_{ij} &\leftarrow \min\{f_{ik} + f_{kj} | k \in B, |f_{ik}|, |f_{kj}| \leq sM\} \\ f_{ij} &\leftarrow \min\{f_{ij}, f'_{ij}\} \end{aligned}$$

für jedes $i, j \in V$.
Für jedes k gilt $f_{ik} + f_{kj} \geq \delta(i,k) + \delta(k,j) \geq \delta(i,j)$
(aus Induktionsvoraussetzung und Dreiecksungleichung)
Also ist der neue Wert von f_{ij} wieder eine obere Schranke für $\delta(i,j)$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

2. Zunächst gilt für alle $i, j \in V$: $f_{ij} = d_{ij}$ und $w_{ij} = 0$ und damit ist die Aussage wahr.
   Wenn f_{ij} ein neuer Wert zugewiesen wird, gilt $1 \leq w_{ij} \leq n$ und $f_{ij} = f_{i,w_{ij}} + f_{w_{ij},j}$.
   Bis zur nächsten Wertzuweisung an f_{ij} wissen wir also, dass gilt $f_{ij} \geq f_{i,w_{ij}} + f_{w_{ij},j}$, da der Wert von f_{ij} sich nicht ändert und die Werte von $f_{i,w_{ij}}$ und $f_{w_{ij},j}$ höchstens kleiner werden können.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

3. Falls die Bedigungen wie angegeben gelten, gilt am Ende der Iteration:

$$f_{ij} \leq f'_{ij} \leq f_{ik} + f_{kj} = \delta(i,k) + \delta(k,j) = \delta(i,j)$$

Da aus der ersten Eigenschaft folgt, dass $f_{ij} \geq \delta(i,j)$, gilt somit $f_{ij} = \delta(i,j)$.

□

Randomisierter Kürzeste-Wege-Algorithmus

Lemma

Sei $s = (3/2)^\ell$ für ein $\ell \in \{1, \ldots, \lceil \log_{3/2} n \rceil\}$.
Dann gilt mit sehr hoher Wahrscheinlichkeit: Wenn es im Graphen einen kürzesten Pfad von i nach j gibt, der höchstens s Kanten benutzt, dann gilt am Ende der Iteration ℓ: $f_{ij} = \delta(i,j)$.

Beweis.

- durch Induktion über ℓ
- Induktionsanfang: Für $\ell = 1$ gilt die Behauptung.
- Induktionsschritt:
  Annahme, die Behauptung gilt für $\ell - 1$,
  zu zeigen: Behauptung gilt auch für ℓ

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

- Seien i und j zwei Knoten, die durch einen kürzesten Pfad p mit höchstens $(3/2)^\ell$ Kanten verbunden sind.
- Falls p höchstens $2s/3$ Kanten hat, dann gilt nach Induktionsvoraussetzung, dass schon nach der $(\ell-1)$-ten Iteration gilt: $f_{ij} = \delta(i,j)$ (mit sehr hoher Wahrscheinlichkeit)
- Also Annahme: p hat mindestens $2s/3$ und höchstens $s = (3/2)^\ell$ Kanten.
- Zunächst zusätzliche Annahme: s ist durch 3 teilbar.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

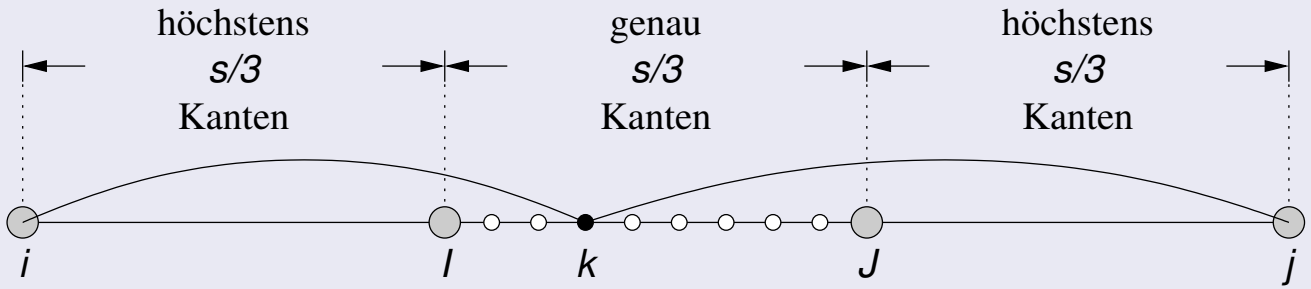


Abbildung: Korrektheitsbeweis für `rand-short-path`

- Seien l und J zwei Knoten auf dem Pfad p, die (auf p) durch exakt $s/3$ Kanten getrennt sind (Solche Knoten muss es aufgrund der Längenbeschränkung $[2s/3, s]$ immer geben).
- Sei A die Menge der Knoten von l bis J (einschließlich) auf p und sei $k \in A$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

- Da k auf einem kürzesten Pfad von i nach j liegt, gilt $\delta(i,j) = \delta(i,k) + \delta(k,j)$.
- Da k zwischen I und J liegt, gibt es kürzeste Pfade von i nach k und von k nach j, die höchstens $2s/3$ Kanten benutzen.
- Aufgrund der Induktionsvoraussetzung wissen wir, dass zu Beginn der ℓ-ten Iteration mit sehr großer Wahrscheinlichkeit gilt: $f_{ik} = \delta(i,k)$ und $f_{kj} = \delta(k,j)$.
- Aus dem dritten Punkt des vorhergehenden Lemmas folgt daher: Falls ein $k \in A \cap B$ existiert (wobei B die gewählte Knotenmenge aus der ℓ-ten Iteration ist), dann gilt am Ende der ℓ-ten Iteration $f_{ij} = \delta(i,j)$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

- Aber wie groß ist die Wahrscheinlichkeit, dass es solch ein $k \in A \cap B$ gibt, dass also $A \cap B \neq \emptyset$ gilt?
- Sei nun $p = \min\{1, (9 \ln n)/s\}$.
- Falls $p = 1$, dann ist $B = V$ und $A \cap B \neq \emptyset$.
- Sei also $p = (9 \ln n)/s < 1$ und jeder Knoten gehört unabhängig mit Wahrscheinlichkeit p zur Teilmenge B.
- Da $|A| \geq s/3$ ist, gilt (mit Hilfe von $1 - x \leq e^{-x}$)

$$\Pr[A \cap B = \emptyset] \leq \left(1 - \frac{9 \ln n}{s}\right)^{s/3} \leq e^{-\frac{9 \ln n}{s} \cdot \frac{s}{3}} = n^{-3}$$

- Da es weniger als n^2 Knotenpaare im Graphen gibt, ist die Fehlerwahrscheinlichkeit für den gesamten Algorithmus kleiner als $n^2 \cdot n^{-3} = 1/n$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

Fix für s ist nicht durch 3 teilbar:

- Definiere die Sequenz $s_0 = 1$ und $s_\ell = \lceil 3s_{\ell-1}/2 \rceil$ für $\ell > 0$.
- Es gilt $s_\ell \geq (3/2)^\ell$.
- Beweis durch Induktion über ℓ, dass (mit hoher Wahrscheinlichkeit) für alle $i, j \in V$ gilt:
  Wenn es einen kürzesten $i \rightsquigarrow j$-Pfad mit $\leq s_\ell$ Kanten gibt, dann gilt am Ende der ℓ-ten Iteration: $f_{ij} = \delta(i,j)$.
- Wenn p ein kürzester $i \rightsquigarrow j$-Pfad mit $\leq s_\ell$ Kanten ist, betrachte Knoten I und J auf p, so dass I und J durch genau $\lfloor s_\ell/2 \rfloor$ Kanten getrennt werden und dass zwischen i und I sowie zwischen J und j höchstens $\lceil s_\ell/2 \rceil$ Kanten liegen.
- Die gleichen Argumente wie im vereinfachten Fall führen zum Beweisabschluss.

Randomisierter Kürzeste-Wege-Algorithmus

- Jedes Paar von Knoten in einem Graphen mit n Knoten (und ohne negative Kreise) ist durch einen kürzesten Pfad mit weniger als n Kanten verbunden (falls ein Pfad existiert).
- Zusammen mit den beiden vorangegangenen Lemmas folgt daraus, dass F nach der letzten Iteration mit sehr hoher Wahrscheinlichkeit die Distanzmatrix ist.
- Weiterhin ist $\delta(i,j) = d_{ij}$ oder w_{ij} liegt auf einem kürzesten Pfad von i nach j.

Randomisierter Kürzeste-Wege-Algorithmus

Lemma

Wenn es keine negativen Kreise im Graph gibt, dann gilt nach der letzten Iteration von `rand-short-path` *mit sehr hoher Wahrscheinlichkeit:*
$\forall i, j \in V$

1. $f_{ij} = \delta(i,j)$
2. *Falls* $w_{ij} = 0$, *dann gilt* $\delta(i,j) = d_{ij}$.
   Sonst gilt $1 \leq w_{ij} \leq n$ *und* $\delta(i,j) = \delta(i,w_{ij}) + \delta(w_{ij},j)$.

Randomisierter Kürzeste-Wege-Algorithmus

Beweis.

1. Folgt aus
   - dem letzten Lemma,
   - dem Fakt, dass in der letzten Iteration gilt $s \geq n$,
   - dem Fakt, dass falls $\delta(i,j) < +\infty$ und falls keine negativen Kreise existieren, es einen kürzesten $i \to j$-Pfad mit höchstens $n-1$ Kanten gibt.
2. Sei nun $f_{ij} = \delta(i,j) < d_{ij}$. Aus dem zweiten Punkt des vorletzten Lemmas folgt, dass nach der letzten Iteration gilt: $1 \leq w_{ij} \leq n$ und $f_{ij} \geq f_{i,w_{ij}} + f_{w_{ij},j}$ bzw. $\delta(i,j) \geq \delta(i,w_{ij}) + \delta(w_{ij},j)$.
   Aus der Dreiecksungleichung folgt: $\delta(i,j) \leq \delta(i,w_{ij}) + \delta(w_{ij},j)$.
   Es gilt also $\delta(i,j) = \delta(i,w_{ij}) + \delta(w_{ij},j)$.

Randomisierter Kürzeste-Wege-Algorithmus

- Man kann auch leicht sehen, dass der Graph genau dann einen Kreis negativer Länge enthält, wenn ein $i \in \{1, \ldots, n\}$ mit $f_{ii} < 0$ existiert.
- Wenn es einen Pfad von i nach j gibt, der durch ein Knoten eines negativen Kreises geht, definieren wir die Distanz von i nach j als $-\infty$.
- Man kann alle solchen Paare in Zeit $\tilde{\mathcal{O}}(n^\omega)$ bestimmen (Galil/Margalit).
- Die von `rand-short-path` zurückgegebene Matrix W enthält eine kompakte Repräsentation von kürzesten Pfaden zwischen allen Knotenpaaren des Graphen.
- Methoden zur Rekonstruktion der kürzesten Pfade werden später beschrieben.

Randomisierter Kürzeste-Wege-Algorithmus

Komplexität von `rand-short-path`:

- Die Zeit der ℓ-ten Iteration wird dominiert durch die Berechnung des Distanzprodukt `dist-prod` einer $n \times m$ mit einer $m \times n$ Matrix, wobei $m \in \mathcal{O}((n \log n)/s)$, mit Betragsbeschränkung sM.
- Eigentlich ist m eine binomialverteilte Zufallsvariable mit $\mathbb{E}[m] \in \mathcal{O}((n \log n)/s)$, was aber an der Analyse nichts ändert.
- Unter der Annahme, dass $s = n^{1-r}$ und $M = n^t$ ist die Zeit für eine Iteration $\tilde{\mathcal{O}}(\min\{n^{t+\omega(1,r,1)+(1-r)}, n^{2+r}\})$.
- Laufzeit einer Iteration ist maximal, wenn $t + \omega(1,r,1) + (1-r) = 2 + r$, bzw. wenn $\omega(1,r,1) = 1 + 2r - t$.
- Da es nur $\mathcal{O}(\log n)$ Iterationen gibt, gilt der folgende Satz.

Randomisierter Kürzeste-Wege-Algorithmus

Satz

Algorithmus `rand-short-path` findet mit sehr hoher Wahrscheinlichkeit alle Distanzen im gegebenen Graph, sowie eine kompakte Repräsentation von kürzesten Pfaden zwischen allen Knotenpaaren.
Wenn der Graph n Knoten hat und die Gewichte alle Integers mit Betrag höchstens $M = n^t$ (mit $t \leq 3 - \omega$) sind, dann ist die Laufzeit $\tilde{\mathcal{O}}(n^{2+\mu(t)})$, wobei für $\mu = \mu(t)$ gilt: $\omega(1, \mu, 1) = 1 + 2\mu - t$.

Randomisierter Kürzeste-Wege-Algorithmus

Bemerkungen:

- Der Ausdruck "mit sehr hoher Wahrscheinlichkeit" im Satz meint eine Wahrscheinlichkeit von mindestens $1 - 1/n$.
- Der Algorithmus kann so verändert werden, dass diese Wahrscheinlichkeit mindestens $1 - n^{-c}$ für eine beliebige Konstante c ist.
- Wenn $M > n^{3-\omega}$ ist, wird keine schnelle Matrixmultiplikation benutzt und die Laufzeit ist $\tilde{\mathcal{O}}(n^3)$.

Komplexität im Fall $M = \mathcal{O}(1)$

- Betrachte Spezialfall $M = \mathcal{O}(1)$, z.B. wenn alle Gewichte aus $\{-1, 0, 1\}$ kommen.
- Algorithmus von Alon, Galil und Margalit braucht in diesem Fall $\tilde{\mathcal{O}}(n^{(3+\omega)/2})$, also ungefähr $\mathcal{O}(n^{2.688})$.
- Laufzeit des neuen Algorithmus von Zwick $\tilde{\mathcal{O}}(n^{2+\mu})$, wobei für μ gilt: $\omega(1, \mu, 1) = 1 + 2\mu$.
- Mit der naiven Schranke $\omega(1, r, 1) \leq 2 + (\omega - 2)r$ erhält man$\mu \leq 1/4 - \omega < 0.616$.
- Mit der besseren Schranke aus dem zweiten Lemma erhält man $\mu \leq (\alpha(\omega - 1) - 1)/(\omega + 2\alpha - 4) < 0.575$.

Komplexität im Fall $M = 1$

Folgerung

Der Algorithmus `rand-short-path` *findet mit sehr großer Wahrscheinlichkeit alle Distanzen und eine kompakte Repräsentation von kürzesten Pfaden zwischen allen Knotenpaaren in einem gerichteten Graph mit n Knoten, in dem alle Kantengewichte aus $\{-1, 0, 1\}$ kommen, in Zeit $\mathcal{O}(n^{2.575})$.*